Linux From Scratch

Version 6.2

Gerard Beekmans

Linux From Scratch: Version 6.2

von Gerard Beekmans

Inhaltsverzeichnis

Einleitung

1. Vorwort

Meine Abenteuer mit Linux begannen 1998 mit dem Herunterladen und Installieren meiner ersten Distribution. Nach einer Weile fielen mir einige Dinge auf, die ich gerne verbessern wollte. Zum Beispiel gefielen mir weder die Zusammenstellung der Bootskripte noch die Voreinstellungen vieler Programme. Ich probierte ein paar alternative Distributionen aus, aber alle hatten neben den Vorteilen auch Nachteile. Schlussendlich wurde mir klar das ich mein eigenes Linux von Grund auf selbst erstellen musste um wirklich zufrieden zu sein.

Im einzelnen bedeutete dies nun, dass ich keinerlei vorkompilierte Pakete, CD-Roms oder Bootdisketten jeglicher Art für die Installation der grundlegenden Werkzeuge verwenden würde. Ich wollte mein bereits laufendes Linux-System als Grundlage benutzen, um darauf mein angepasstes Linux zu entwickeln. Dieses „perfekte" Linux-System sollte die Stärken der verschiedenen Distributionen ohne deren Schwächen vereinen. Zu Beginn war die Umsetzung der Idee ziemlich entmutigend. Aber ich blieb engagiert bei der Sache. Ich wollte schließlich ein Linux-System, das meinen Ansprüchen gerecht wurde, anstatt einer Standard-Distribution, die nicht meinen Wünschen entsprach.

Um das meinen Wünschen entsprechende Linux zu erstellen musste ich erstmal viele Probleme mit gegenseitigen Abhängigkeiten und jede Menge Kompilierfehler beheben. Als ich damit fertig war, hatte ich jedoch ein voll funktionsfähiges und anpassbares Betriebssystem. Meine Vorgehensweise ermöglicht das Erstellen sehr kompakter Linux-Systeme, die schneller sind und weniger Speicher verbrauchen als viele herkömmliche Betriebssysteme. Ich nannte dieses System Linux From Scratch, oder einfach kurz LFS.

Ich teilte meine Erfahrungen mit anderen Mitgliedern der Linux-Gemeinschaft und es stellte sich schnell ein wachsendes Interesse an der Fortsetzung meiner Arbeit mit Linux heraus: Selbstgebaute LFS-Systeme entsprechen nicht einfach nur Spezifikationen und Anforderungen von Anwendern, sondern sind auch eine ideale Lernbasis für Programmierer und Systemadministratoren mit der man seine Linux-Kenntnisse erweitern kann. Aus diesem breiten Interesse heraus entstand dann das Projekt Linux From Scratch.

Das Buch *Linux From Scratch* vermittelt dem Leser das Wissen und nötige Anleitungen um ein eigenes Linux-System zu entwerfen und zu erstellen. Es hebt das Projekt Linux From Scratch und die Vorteile dieses Systems hervor. Der Leser kann alle Eigenschaften des Systems selber vorgeben, inklusive dem Layout der Ordnerstruktur, Skript-Einstellungen und Sicherheit. Das System wird direkt aus dem Quellcode kompiliert und man kann selber entscheiden, wo, warum und wie Programme installiert werden. Dieses Buch ermöglicht es jedem, Linux-Systeme an die eigenen Bedürfnisse anzupassen und mehr Kontrolle über das System zu erlangen.

Ich wünsche Ihnen viel Freude bei der Arbeit an Ihrem eigenen LFS-System. Genießen Sie die Vorteile eines Systems, das wirklich *Ihr Eigen* ist.

--
Gerard Beekmans
gerard@linuxfromscratch.org

2. Warum sollte man dieses Buch lesen?

Es gibt viele gute Gründe, dieses Buch zu lesen. Die meisten Leser möchten lernen, wie man ein Linux-System direkt aus den Quellen erstellt. Oft wird die Frage gestellt: „Warum soll man sich die Mühe machen, ein Linux-System selbst zu erstellen, wenn man einfach ein fertiges Linux herunterladen und installieren kann?“. Das ist eine berechtigte Frage und gleichzeitig auch der Anstoß für dieses Kapitel.

Ein wichtiges Ziel von LFS ist, dem Leser beizubringen wie Linux intern funktioniert. Der Selbstbau eines Linux-Systems veranschaulicht Ihnen, was Linux seinen Herzschlag verleiht und wie die Komponenten zusammenarbeiten und voneinander abhängen. Das Beste daran ist, dass Sie durch den Lernprozess in die Lage versetzt werden, Linux an Ihre eigenen Anforderungen und Vorlieben anzupassen.

Einer der grössten Vorteile von LFS ist, dass Sie mehr Kontrolle über Ihr System erhalten, ohne sich auf die Linux-Version von jemand anders verlassen zu müssen. Mit LFS sitzen *Sie selbst* am Steuer und können jeden Aspekt Ihres Systems beeinflussen, wie zum Beispiel das Ordner-Layout oder die Einrichtung der Bootskripte. Auch bestimmen Sie, wo, warum und wie Programme installiert werden.

Ein weiterer Vorteil von LFS ist die Möglichkeit, Linux sehr kompakt zu halten. Wenn Sie eine übliche Linux-Distribution verwenden, installieren Sie für gewöhnlich viele Programme die Sie nie benutzen werden. Diese liegen dann unnütz auf der Festplatte und verbrauchen Speicherplatz (oder CPU-Ressourcen). Es ist leicht, ein LFS-System unter 100 MB zu installieren. Das ist immer noch zu groß? Einige LFS-Mitglieder haben an einem sehr kleinen Embedded-Linux gearbeitet. Sie haben einen Apache-Webserver auf einem Linux From Scratch mit gerade mal 8 MB belegtem Festplattenspeicher installiert. Durch weitere Einschränkungen könnte das System auf bis zu 5 MB oder weniger schrumpfen. Versuchen Sie das mal mit einer herkömmlichen Linux-Distribution.

Man könnte die verschiedenen Linux-Distributionen mit einem Hamburger aus einer Fast-Food-Kette vergleichen—man weiß nie genau was man isst. LFS auf der anderen Seite wäre nicht der Burger, sondern vielmehr das Rezept. Man kann das Rezept überprüfen, ungewollte Zutaten weglassen und eigene Zutaten nach Geschmack und Belieben hinzufügen. Wenn man zufrieden ist bereitet man es zu. Und auch hier kann man variieren—braten, backen, tiefgefrieren, grillen oder roh essen, ganz wie man will.

Es gibt noch weitere Analogien: Vergleichen Sie LFS z. B. mit einem Fertighaus. LFS wäre in dem Fall der Plan für den Grundriss, aber bauen müssen Sie das Haus selber. Jeder kann den Plan ganz nach Belieben ändern.

Nicht zuletzt ist auch Sicherheit ein Vorteil eines selbstgebauten Linux-Systems. Wer ein Linux-System selber aus den Quellen kompiliert, kann sämtliche Quelltexte sichten und alle für wichtig erachteten Sicherheitspatches installieren. Man muss nicht warten bis jemand anders Binärpakete zur Behebung von Sicherheitslöchern bereitstellt. Solange Sie die Patches nicht selber prüfen und installieren, ist auch nicht sichergestellt, dass das Binärpaket korrekt kompiliert wurde und dass es das Problem auch wirklich behebt.

Das erklärte Ziel von Linux From Scratch ist, ein vollständiges, lauffähiges und grundsolides System zu erstellen. Wenn Sie nur interessiert, was genau beim Hochfahren Ihres Computers geschieht, dann empfehlen wir das HOWTO „From Power Up To Bash Prompt“; Sie bekommen es unter *http://axiom.anu.edu.au/~okeefe/p2b/* oder auf der Webseite des Linux Documentation Project unter *http://www.tldp.org/HOWTO/From-PowerUp-To-Bash-Prompt-HOWTO.html*. Mit Hilfe dieses HOWTOs wird ein blankes System installiert, das dem in diesem Buch sehr ähnlich ist, sich aber ausschließlich auf das Erstellen eines Systems konzentriert, das eine Bash-Shell booten kann. Halten Sie sich am besten Ihr Ziel vor Augen: wenn Sie Linux installieren und nebenbei dazulernen möchten, dann ist Linux From Scratch für Sie geeignet.

Es gibt einfach zu viele gute Gründe für das Erstellen eines eigenen LFS-Systems um sie hier alle aufzuzählen, die hier genannten Gründe sind nur die Spitze des Eisberges. Während Sie mit LFS arbeiten und Erfahrungen sammeln, werden Sie selbst schnell feststellen wie wertvoll Informationen und Wissen

über Linux sind.

3. Voraussetzungen

Der Selbstbau eines LFS ist keine leichte Aufgabe. Man benötigt ein entsprechendes Vorwissen zur Administration von Unix-Systemen, sonst fällt es schwer, bestimmte Kommandos zu verstehen oder auf Fehlersuche zu gehen. Sie als Leser sollten als absolutes Minimum zumindest mit der Kommandozeile (Shell) umgehen können (dazu gehört das Kopieren und Verschieben von Dateien und Ordnern, Auflisten von Ordner- und Dateiinhalte und das Wechseln des aktuellen Ordners). Außerdem setzen wir voraus, dass Sie grundsätzlich wissen, wie man Linux-Software benutzt und installiert.

Weil das LFS-Buch dieses Vorwissen als *absolutes Minimum* voraussetzt, werden Sie in den verschiedenen LFS Support-Foren höchstwahrscheinlich keine Hilfe bekommen, wenn Sie Fragen ohne das notwendige Basiswissen stellen. Möglicherweise bleiben Ihre Fragen einfach nur unbeantwortet oder man verweist Sie auf diesen Text.

Bevor Sie ein LFS-System erstellen, lesen Sie bitte die folgenden HOWTOs:

- Software-Building-HOWTO *http://www.tldp.org/HOWTO/Software-Building-HOWTO.html*

  Das Software-Building-HOWTO ist ein umfangreiches Handbuch zum Erstellen und Installieren „allgemeiner“ UNIX-Software unter Linux.

- The Linux Users' Guide *http://www.linuxhq.com/guides/LUG/guide.html*

  Dieses Handbuch erklärt die Verwendung ausgewählter Linux-Software.

- The Essential Pre-Reading Hint *http://www.linuxfromscratch.org/hints/downloads/files/essential_prereading.tx*

  Dies ist eine LFS-Anleitung, die speziell für neue Linux-Anwender geschrieben wurde. Sie enthält eine Linksammlung sehr guter Informationsquellen zu allen möglichen Themen. Jeder der LFS installieren möchte, sollte zumindest den Großteil der dort behandelten Themen verstehen.

4. Mindestanforderungen an das Host-System

Ihr Host-System sollte über die folgende Software mit den angegebenen Minimalversionen verfügen. Für die meisten modernen Linux-Distributionen sollte dies kein Problem darstellen. Bitte beachten Sie allerdings, dass die meisten Distributionen die Header-Dateien zu Programmen in extra-Pakete packen, meist mit Namen wie „<Paketname>-devel“ oder „<Paketname>-dev“. Bitte stellen Sie sicher, dass Sie auch diese Pakete mit den Headern installiert haben.

- **Bash-2.05a**
- **Binutils-2.12** (Versionen größer 2.16.1 werden nicht empfohlen, weil sie nicht getestet wurden)
- **Bzip2-1.0.2**
- **Coreutils-5.0** (oder Sh-Utils-2.0, Textutils-2.0 und Fileutils-4.1)
- **Diffutils-2.8**
- **Findutils-4.1.20**
- **Gawk-3.0**
- **Gcc-2.95.3** (Versionen größer 4.0.3 werden nicht empfohlen, weil sie nicht getestet wurden.)
- **Glibc-2.2.5** (Versionen größer 2.3.6 werden nicht empfohlen, weil sie nicht getestet wurden.)
- **Grep-2.5**
- **Gzip-1.2.4**
- **Linux-Kernel-2.6.x** (wurde mit GCC-3.0 oder neuer kompiliert)

  Der Grund für diese Kernelanforderung liegt darin, dass die Unterstützung für thread-local storage in Binutils nicht einkompiliert wird und die Native POSIX Threading-Bibliothek (NPTL) abstürzt, wenn der Host-Kernel nicht mindestens Version 2.6.x ist und mit GCC 3.0 oder neuer kompiliert wurde.Kern

  Wenn der Host-Kernel jünger als 2.6.x ist oder dass er nicht mit mindestens GCC 3.0 (oder neuer) kompiliert wurde, dann muss auf dem Host erstmal ein solcher Kernel installiert und gebootet werden. Es gibt zwei Möglichkeiten, das Problem zu beheben: Überprüfen Sie, ob der Hersteller Ihrer Host-Distribution einen entsprechenden Kernel zur Verfügung stellt. Wenn ja, installieren Sie diesen. Falls der Hersteller jedoch keinen passenden Kernel ausliefert (oder Sie diesen aus irgendwelchen Gründen nicht installieren möchten), dann können Sie selbst einen 2.6er Kernel kompilieren. Eine Hilfestellung dazu finden Sie in Kapitel 8 (vorausgesetzt der Host verwendet GRUB als Bootloader).
- **Make-3.79.1**
- **Patch-2.5.4**
- **Sed-3.0.2**
- **Tar-1.14**

Um herauszufinden, ob Ihr Host-System alle notwendigen Programmversionen installiert hat, führen Sie den folgenden Befehl aus:

```
cat > version-check.sh << "EOF"
#!/bin/bash

# Einfaches Skript zum Auflisten der Versionsnummern wichtiger Werkzeuge

bash --version | head -n1 | cut -d" " -f2-4
echo -n "Binutils: "; ld --version | head -n1 | cut -d" " -f3-4
bzip2 --version 2>&1 < /dev/null | head -n1 | cut -d" " -f1,6-
echo -n "Coreutils: "; chown --version | head -n1 | cut -d")" -f2
diff --version | head -n1
find --version | head -n1
gawk --version | head -n1
gcc --version | head -n1
/lib/libc.so.6 | head -n1 | cut -d" " -f1-7
grep --version | head -n1
gzip --version | head -n1
cat /proc/version | head -n1 | cut -d" " -f1-3,5-7
make --version | head -n1
patch --version | head -n1
sed --version | head -n1
tar --version | head -n1

EOF

bash version-check.sh
```

5. Konventionen in diesem Buch

Das Buch hält sich an einige typografische Konventionen, die zum allgemein besseren Verständnis beitragen sollen. Es folgen einige Beispiele:

```
./configure --prefix=/usr
```

Solange nicht anders angegeben, muss Text in dieser Textform exakt so eingegeben werden, wie er zu sehen ist. Diese Darstellung wird auch in den Erklärungstexten verwendet, um sich eindeutig auf bestimmte Kommandos zu beziehen.

```
install-info: unknown option '--dir-file=/mnt/lfs/usr/info/dir'
```

Diese Textform (Text mit fester Zeichenbreite) stellt Bildschirmausgaben dar. Text in dieser Form ist oft die Ausgabe von vorher eingegebenen Kommandos. Außerdem wird diese Textform für Dateinamen wie z. B. `/etc/ld.so.conf` verwendet.

Hervorhebung

Diese Textform wird für verschiedene Zwecke benutzt und soll wichtige Details hervorheben.

http://www.linuxfromscratch.org/

Auf diese Weise werden Links dargestellt, sowohl innerhalb des Buches als auch zu externen Seiten wie z. B. HOWTOs, Download-Adressen und Webseiten.

```
cat > $LFS/etc/group << "EOF"
root:x:0:
bin:x:1:
......
EOF
```

Solche Textabschnitte werden hauptsächlich beim Erstellen von Konfigurationsdateien verwendet. Der obige Block erzeugt die Datei `$LFS/etc/group` mit dem nachfolgenden Inhalt bis die Zeichenfolge EOF erkannt wird. Normalerweise müssen Sie Text in dieser Form exakt so eingeben wie er zu sehen ist.

`<ZU ERSETZENDER TEXT>`

Dies ist Text, den Sie nicht einfach blindlings abschreiben oder kopieren und einfügen können.

`[OPTIONALER TEXT]`

Mit diesen Klammern wird Text markiert, der option ist.

`passwd(5)`

Diese Textform wird verwendet, um sich auf eine Man-page zu beziehen. Die Zahl in Klammern bezeichnet eine bestimmte Sektion in **man**. **passwd** z. B. hat zwei Man-pages. Nach der LFS-Anleitung werden diese nach `/usr/share/man/man1/passwd.1` und `/usr/share/man/man5/passwd.5` installiert. Beide Man-pages enthalten unterschiedliche Informationen und Themenbereiche. Wenn Sie also `passwd(5)` lesen, bezieht sich das Buch explizit auf `/usr/share/man/man5/passwd.5`. Das Kommando **man passwd** zeigt die erste gefundene Man-page zu passwd an. Das wäre in diesem Fall `/usr/share/man/man1/passwd.1`. Um in diesem Beispiel die richtige Man-page aus Sektion 5 anzuzeigen müssen Sie das Kommando **man 5 passwd** verwenden. Die meisten Man-pages haben keine doppelten Seiten-Namen in unterschiedlichen Sektionen, daher ist **man *`<Programmname>`*** meistens ausreichend.

6. Aufbau

Das Buch ist in die folgenden Abschnitte unterteilt.

6.1. Teil I - Einführung

Teil I erläutert einige wichtige Hinweise zur Installation und schafft Grundlagen zur allgemeinen Verwendung des Buches.

6.2. Teil II - Vorbereitungen zur Installation

Teil II bereitet den eigentlichen Installationsvorgang vor—Anlegen einer Partition, Herunterladen der Pakete und Kompilieren der temporären Werkzeuge.

6.3. Teil III - Installation

Teil III führt Sie Schritt für Schritt durch die eigentliche Installation von LFS—Kompilieren und Installieren aller Pakete, Aufsetzen der Bootskripte und Installieren des Kernels. Das resultierende Linux-System ist die Basis, auf der später weitere Software installiert wird und auf der das System ganz nach Ihrem Belieben erweitert werden kann. Am Ende des Buches finden Sie zu Referenzzwecken eine Liste aller Programme, Bibliotheken und wichtiger Dateien, die während der Arbeit mit diesem Buch installiert wurden.

7. Errata

Die für LFS verwendete Software wird laufend aktualisiert und erweitert. Nach Erscheinen des Buches könnten Sicherheitshinweise und Fehlerbereinigungen hinzugekommen sein. Bevor Sie mit dem Bau von LFS beginnen, sollten Sie unter *http://www.linuxfromscratch.org/lfs/errata/6.2/* nachsehen, ob Änderungen an den Installationsanleitungen oder an Softwareversionen nötig sind. Bitte notieren Sie alle nötigen Änderungen und wenden Sie diese in den entsprechenden Kapiteln an.

Teil I. Einführung

Kapitel 1. Einführung

1.1. Vorgehensweise zur Installation von LFS

Sie werden LFS mit Hilfe einer bereits laufenden Linux-Distribution (wie z. B. Debian, Mandriva, Red Hat oder SuSE) installieren. Das bestehende System (der Host) wird als Einstiegspunkt benutzt, denn Sie brauchen Programme wie Compiler, Linker und eine Shell, um Ihr neues System zu erstellen. Normalerweise sind alle notwendigen Programme bereits installiert, wenn Sie bei der Installation Ihrer Distribution die Kategorie „Entwicklung“ ausgewählt haben.

Falls Sie nur wegen LFS kein neues Host-System installieren möchten dann sollten Sie die Linux From Scratch Live-CD verwenden. Die CD enthält ein voll funktionsfähiges Host-System mit allen notwendigen Werkzeugen für eine erfolgreiche Installation. Zudem enthält sie auch alle Quellpakete, Patches und eine Online-Version dieses Buches. Wenn Sie die CD verwenden, brauchen Sie auch keine Netzwerkverbindung weil nichts mehr heruntergeladen werden muss. Weitere Informationen zu der CD finden Sie unter *http://www.linuxfromscratch.org/livecd/*. Dort können Sie auch eine Kopie der CD herunterladen.

Kapitel 2 beschreibt das Anlegen einer neuen Linux-Partition und eines Dateisystems, auf dem Ihr neues LFS-System kompiliert und installiert wird. In Kapitel 3 erfahren Sie, welche Pakete und Patches Sie herunterladen müssen. Kapitel 4 erklärt das Einrichten einer funktionsfähigen Arbeitsumgebung für die kommenden Arbeitsschritte. Bitte lesen Sie Kapitel 4 aufmerksam durch! Es behandelt ein paar mögliche Schwierigkeiten, die Ihnen vor der Arbeit mit Kapitel 5 und den folgenden bekannt sein sollten.

Kapitel 5 beschreibt dann die Installation der Pakete für die grundlegende Entwicklungsumgebung (im weiteren Verlauf des Buches *Toolchain* genannt). Die Toolchain ist eine Sammlung der nötigsten Werkzeuge und wird später in Kapitel 6 verwendet um das endgültige System zu erstellen. Einige dieser Pakete werden zum Auflösen rekursiver Abhängigkeiten benötigt—zum Beispiel brauchen Sie einen Compiler, um einen Compiler zu kompilieren.

Kapitel 5 erklärt auch, wie die erste Version der Basiswerkzeuge, inklusive Binutils und GCC, erzeugt wird. „Erste Version“ bedeutet in diesem Zusammenhang, dass diese zwei Pakete installiert werden. Als zweiten Schritt kompilieren Sie Glibc, die C-Bibliothek. Glibc wird mit den Programmen der im ersten Schritt erstellten Basiswerkzeuge kompiliert. Im dritten Schritt erstellen Sie dann die zweite Version der Basiswerkzeuge. Sie linken die Programme dynamisch gegen die gerade frisch installierte Glibc. Die verbleibenden Pakete aus Kapitel 5 werden alle diesen zweiten Durchlauf der Toolchain verwenden und dynamisch gegen die neue, hostunabhängige Glibc gelinkt. Wenn dies erledigt ist, ist der weitere Installationsvorgang — mit Ausnahme des Kernels — nicht mehr von der Linux-Distribution auf dem Host-System abhängig.

Dies scheint erstmal eine Menge Arbeit zu sein um sich von der Host-Distribution zu lösen. Eine vollständige Erklärung finden Sie in Abschnitt 5.2, „Technische Anmerkungen zur Toolchain“.

In Kapitel 6 wird das endgültige LFS-System erstellt. Wir benutzen das Programm **chroot** (chroot = change root = wechseln der Basis), um eine Shell in einer virtuellen Umgebung zu starten. In der neuen Shell ist der Basisordner auf die LFS-Partition eingestellt. Chroot'en ist so ähnlich wie Neustarten und Einhängen der LFS-Partition als root-Dateisystem. Das Erstellen eines bootfähigen Systems würde allerdings zusätzliche Arbeit erfordern und ist an dieser Stelle absolut unnötig. Außerdem hat chroot'en den Vorteil, dass Sie das Host-Betriebssystem weiter nebenher verwenden können während Sie in der Shell das LFS installieren. Während Sie also warten bis alle Pakete kompiliert sind, können Sie einfach auf ein anderes VT (Virtuelles Terminal) oder auf den X-Desktop wechseln und dort wie gewohnt weiterarbeiten.

Zum Abschluss der Installation werden in Kapitel 7 die Bootskripte eingerichtet; der Kernel sowie der Bootloader werden in Kapitel 8 behandelt. Wenn Sie das Buch zuende gelesen haben finden Sie in Kapitel 9 Links auf weiterführende Hilfeseiten. Abschließend ist der Computer bereit für einen Neustart mit dem

neuen LFS-System.

Nun kennen Sie die allgemeine Vorgehensweise in stark zusammengefasster Form. Die jeweiligen Kapitel beinhalten natürlich detailliertere Informationen. Machen Sie sich keine Gedanken, falls hier noch etwas unklar sein sollte; alle offene Fragen werden sich im weiteren Verlauf klären.

1.2. Ressourcen

1.2.1. FAQ

Wenn Sie beim Erstellen von LFS Schwierigkeiten oder Fragen haben oder wenn Sie einen (Rechtschreib-) Fehler im Buch finden, dann lesen Sie bitte die FAQ (Frequently Asked Questions - häufig gestellte Fragen) unter *http://www.linuxfromscratch.org/faq/*.

1.2.2. Mailinglisten

Auf dem Server `linuxfromscratch.org` werden einige Mailinglisten für die Entwicklung des LFS-Projektes betrieben. Unter anderem befinden sich dort auch die Entwickler- und Support-Mailinglisten. Falls die FAQ Ihnen mit Ihrem Problem nicht helfen kann, sollten Sie im nächsten Schritt die Mailinglisten unter *http://www.linuxfromscratch.org/search.html* durchsehen.

Welche Listen es gibt, wie Sie eine Liste abonnieren können, wo Sie die Archive finden und vieles mehr erfahren Sie unter *http://www.linuxfromscratch.org/mail.html*.

1.2.3. IRC

Viele Mitglieder aus der LFS-Gemeinschaft bieten ihre Hilfe über unseren IRC-Server an. Bevor Sie hier Hilfe suchen lesen Sie bitte zumindest die FAQ und die Archive unserer Mailinglisten und suchen dort nach einer Antwort auf Ihre Frage. Der IRC-Server hat die Adresse `irc.linuxfromscratch.org`. Der Support-Chatraum heißt #LFS-support.

1.2.4. Referenzen

Weitere Informationen und nützliche Tipps zu Software-Paketen finden Sie in der LFS Paket-Referenz unter *http://www.linuxfromscratch.org/~matthew/LFS-references.html*.

1.2.5. Softwarespiegel

Das LFS-Projekt hat viele über die ganze Welt verteilte Softwarespiegel. Diese stellen die Website zur Verfügung und vereinfachen das Herunterladen der benötigten Programme. Bitte besuchen Sie *http://www.linuxfromscratch.org/mirrors.html*, dort können Sie eine Liste der aktuellen Softwarespiegel einsehen.

1.2.6. Kontakt

Bitte senden Sie alle Fragen und Kommentare direkt an eine der LFS-Mailinglisten (siehe oben).

1.3. Hilfe

Wenn Sie beim Lesen des Buches auf ein Problem stoßen, sollten Sie als erstes in der FAQ unter *http://www.linuxfromscratch.org/faq/#generalfaq* nachlesen—die meisten Fragen werden hier schon beantwortet. Falls nicht, versuchen Sie die Ursache des Problems zu finden. Die folgende Anleitung könnte Ihnen bei der Fehlersuche behilflich sein: *http://www.linuxfromscratch.org/hints/downloads/files/errors.txt*.

Falls Sie Ihr Problem nicht in der FAQ finden, dann durchsuchen Sie am besten die Mailinglisten unter *http://www.linuxfromscratch.org/search.html*.

Wenn das nicht hilft, ist man im Internet Relay Chat (IRC) und auf den Mailinglisten (Abschnitt 1.2, „Ressourcen“) gern bereit, Ihnen zu helfen. Allerdings erhalten wir jeden Tag viele Anfragen, die durch einfaches Lesen der FAQ oder Durchlesen der Mailinglisten beantwortet werden könnten. Wir können Ihnen am besten helfen, wenn Sie zuerst selbst ein wenig auf Fehlersuche gehen. Dadurch können wir uns besser auf die wirklich schwierigen Fragen konzentrieren. Wenn Ihre eigenen Recherchen keine Ergebnisse zutage bringen, dann unterstützen Sie uns bitte, indem Sie alle relevanten Informationen direkt mitsenden.

1.3.1. Dinge, die Sie angeben sollten

Neben einer kurzen Zusammenfassung des Problems ist es wichtig, dass Sie uns noch folgende Dinge mitteilen:

- Die Version dieses Buches (in diesem Fall Version 6.2),
- Host-Distribution und -Versionsnummer, die Sie zur Installation von LFS verwenden,
- die Software oder der Abschnitt, der Ihnen Probleme bereitet,
- die exakte Fehlermeldung bzw. die genauen Symptome, die Sie sehen,
- ob Sie von den Anleitungen im Buch abgewichen sind, und wenn ja, wie.



Anmerkung

Beachten Sie: Wir werden Ihnen auch helfen wenn Sie von den Anleitungen im Buch abgewichen sind. Schließlich ist die freie Wahl ein wichtiger Grundsatz von LFS. Ihr Hinweis hilft uns lediglich, die möglichen Ursachen für Ihr Problem besser zu erkennen.

1.3.2. Probleme mit configure-Skripten

Wenn beim Durchlaufen der **configure**-Skripte ein Problem auftritt, schauen Sie bitte zuerst in die Datei `config.log`. Sie enthält Fehlermeldungen, die auf dem Bildschirm normalerweise nicht angezeigt werden. Geben Sie die *relevanten* Fehlermeldungen mit an, wenn Sie um Hilfe bitten.

1.3.3. Kompilierprobleme

Sowohl Bildschirmausgaben als auch der Inhalt einiger Dateien sind für uns nützlich, um Ihnen bei der Fehlersuche zu helfen. Die Ausgaben des **configure**-Skriptes und die des Befehls **make** können sehr hilfreich sein. Bitte kopieren Sie aber nicht einfach blindlings die gesamte Ausgabe; andererseits sollte es aber auch nicht zu wenig sein. Als Beispiel für sinnvolle Informationen soll Ihnen folgende Ausgabe von **make** helfen:

```
gcc -DALIASPATH=\"/mnt/lfs/usr/share/locale:.\"
-DLOCALEDIR=\"/mnt/lfs/usr/share/locale\"
-DLIBDIR=\"/mnt/lfs/usr/lib\"
-DINCLUDEDIR=\"/mnt/lfs/usr/include\" -DHAVE_CONFIG_H -I. -I.
-g -O2 -c getopt1.c
gcc -g -O2 -static -o make ar.o arscan.o commands.o dir.o
expand.o file.o function.o getopt.o implicit.o job.o main.o
misc.o read.o remake.o rule.o signame.o variable.o vpath.o
default.o remote-stub.o version.o opt1.o
-lutil job.o: In function `load_too_high':
/lfs/tmp/make-3.79.1/job.c:1565: undefined reference
to `getloadavg'
collect2: ld returned 1 exit status
make[2]: *** [make] Error 1
make[2]: Leaving directory `/lfs/tmp/make-3.79.1'
make[1]: *** [all-recursive] Error 1
make[1]: Leaving directory `/lfs/tmp/make-3.79.1'
make: *** [all-recursive-am] Error 2
```

In diesem Fallbeispiel kopieren viele leider nur den unteren Teil:

```
make [2]: *** [make] Error 1
```

Das reicht uns aber nicht, um Ihnen bei der Fehlerdiagnose helfen zu können, denn es sagt uns nur, *dass* etwas schiefgelaufen ist, aber nicht *was*. Sie müssen den ganzen oben gezeigten Block angeben, denn er enthält das ausgeführte Kommando und die dazugehörige Fehlermeldung(en).

Eric S. Raymond hat zu diesem Thema einen sehr guten Artikel geschrieben. Sie finden ihn unter *http://catb.org/~esr/faqs/smart-questions.html*. Lesen und befolgen Sie bitte seine Tipps. So erhöhen Sie Ihre Chance, dass Sie auf Ihre Frage eine Antwort erhalten, mit der Sie auch etwas anfangen können.

Teil II. Vorbereitungen zur Installation

Kapitel 2. Vorbereiten einer neuen Partition

2.1. Einführung

In diesem Kapitel bereiten Sie die Partition vor, die später Ihr neues LFS-System enthalten wird. Sie erstellen die Partition, erzeugen darauf ein Dateisystem und hängen sie anschließend ein (mounten).

2.2. Erstellen einer neuen Partition

Wie die meisten Betriebssysteme wird auch LFS auf einer separaten Partition installiert. Sie sollten für LFS bereits eine leere Partition haben, oder eine neue Partition anlegen. Ein LFS kann aber auch in einer bereits belegten Partition installiert werden, sodass mehrere Betriebssyteme nebeneinander existieren. Das Dokument *http://www.linuxfromscratch.org/hints/downloads/files/lfs_next_to_existing_systems.txt* erklärt, wie man dies einrichtet. Im Buch gehen wir allerdings nur darauf ein, wie man LFS auf eine leere, dedizierte Partition installiert.

Für ein Minimal-System benötigen Sie eine Partition mit etwa 1,3 GB Platz. Das reicht aus, um die Quellpakete zu speichern und alle Pakete zu installieren. Wenn Sie Ihr LFS später als primäres Betriebssystem nutzen möchten, brauchen Sie zum Nachinstallieren weiterer Pakete mehr Platz (ca. 2 bis 3 GB). Das LFS-System selbst benötigt selbstverständlich nicht so viel Speicher. Der größte Teil wird als temporärer Speicher benötigt: Das Kompilieren von Paketen kann eine Menge Festplattenplatz in Anspruch nehmen, der aber nach dem Kompiliervorgang wieder freigegeben wird.

Manchmal ist zu wenig Random-Access-Memory (RAM, Arbeitsspeicher) verfügbar, daher sollte man eine kleine Partition als `Swap`-Partition einrichten—das ist Speicherplatz, den der Kernel zum Auslagern selten genutzter Daten verwendet. Das schafft Platz im Arbeitsspeicher für wichtigere Dinge. Die `Swap`-Partition in Ihrem LFS kann dieselbe sein wie die, die Sie bereits für ihr Host-System nutzen. Falls Sie also schon eine funktionsfähige Swap-Partition haben, müssen Sie keine zusätzliche erstellen.

Rufen Sie ein Partitionierungsprogramm wie zum Beispiel **cfdisk** oder **fdisk** auf. Als Argument übergeben Sie die Festplatte, auf der Sie die neue Partition erstellen möchten—zum Beispiel `/dev/hda` für die primäre Integrated Drive Electronics (IDE) Festplatte. Erstellen Sie eine native Linux-Partition (und eine `Swap`-Partition falls nötig). Bitte lesen Sie die Man-Page zu **cfdisk** oder **fdisk**, wenn Ihnen die Bedienung dieser Programme unklar ist.

Merken Sie sich die Bezeichnung Ihrer neuen Partition — sie könnte `hda5` oder ähnlich lauten. Das Buch bezeichnet diese Partition im weiteren Verlauf als die LFS-Partition. Wenn Sie (nun) eine `Swap`-Partition haben, merken Sie sich auch deren Bezeichnung. Sie werden sie später in die Datei `/etc/fstab` eintragen.

2.3. Erstellen eines Dateisystems auf der neuen Partition

Nun haben Sie eine leere Partition und können darauf ein Dateisystem anlegen. Das meistverbreitete Dateisystem unter Linux ist das Second Extended Filesystem (`ext2`); aber im Zuge der heute üblichen großen Festplatten gewinnen Journal-Dateisysteme immer mehr an Beliebtheit. Das `ext3`-Dateisystem ist eine weit verbreitete Erweiterung von `ext2` und kompatibel mit den E2fsprogs. An dieser Stelle erzeugen wir ein `ext3`-Dateisystem. Unter *http://www.linuxfromscratch.org/blfs/view/svn/postlfs/filesystems.html* finden Sie Anleitungen zum Einrichten anderer Dateisysteme.

Zum Erzeugen eines `ext3`-Dateisystems auf der LFS-Partition führen Sie bitte das folgende Kommando aus:

```
mke2fs -jv /dev/<xxx>
```

Ersetzen Sie *<xxx>* durch den Namen der LFS-Partition (wie zum Beispiel `hda5`).



Anmerkung

Einige Distributionen haben Zusatzfunktionen in ihre Werkzeuge zum Erzeugen von Dateisystemen (E2fsprogs) eingebaut. Dies kann später beim Booten Ihres neuen LFS zu Probleme führen, weil diese Erweiterungen in den von LFS installierten E2fsprogs nicht installiert sind. Sie könnten z. B. eine Fehlermeldung wie „unsupported filesystem features; upgrade your e2fsprogs“ erhalten. Mit dem folgenden Kommando können Sie herausfinden, ob Ihr Host-System solche zusätzlichen Funktionen verwendet:

```
debugfs -R feature /dev/<xxx>
```

Wenn die Ausgabe mehr Funktionen als `has_journal`, `dir_index`, `filetype`, `large_file`, `resize_inode`, `sparse_super` oder `needs_recovery` enthält, dann sind in Ihrem Host-System zusätzliche Erweiterungen installiert. Sie sollten spätere Probleme vermeiden indem Sie das normale Paket E2fsprogs kompilieren und die daraus resultierenden Programme zum Erzeugen des Dateisystems auf Ihrer LFS-Partition verwenden:

```
cd /tmp
tar -xjvf /Pfad/zu/den/Quellen/von/e2fsprogs-1.39.tar.bz2
cd e2fsprogs-1.39
mkdir -v build
cd build
../configure
make #ANMERKUNG: Führen Sie bitte _nicht_ 'make install' aus!
./misc/mke2fs -jv /dev/<xxx>
cd /tmp
rm -rfv e2fsprogs-1.39
```

Wenn Sie eine `Swap`-Partition erstellt haben, müssen Sie diese mit dem untenstehenden Befehl initialisieren (dies bezeichnet man auch als formatieren). Wenn Sie eine bereits existierende `Swap`-Partition verwenden, muss diese nicht initialisiert werden.

```
mkswap /dev/<yyy>
```

Bitte ersetzen Sie *<yyy>* durch den Namen Ihrer `Swap`-Partition.

2.4. Einhängen (mounten) der neuen Partition

Nachdem Sie nun ein Dateisystem erzeugt haben, sollten Sie natürlich auch darauf zugreifen können. Dazu müssen Sie erst einen Mountpunkt wählen und es dann dort einhängen (mounten). Wir gehen davon aus, dass das Dateisystem unter `/mnt/lfs` eingehängt wird. Sie können sich aber auch jeden anderen Ordner aussuchen.

Wählen Sie nun einen Mountpunkt und tragen Sie ihn in die Umgebungsvariable `LFS` ein. Dazu können Sie diesen Befehl verwenden:

```
export LFS=/mnt/lfs
```

Als nächstes erzeugen Sie den Ordner den Sie als Mountpunkt gewählt haben und hängen das LFS-Dateisystem ein:

```
mkdir -pv $LFS
mount -v -t ext3 /dev/<xxx> $LFS
```

Bitte setzen Sie statt *<xxx>* die Bezeichnung der LFS-Partition ein.

Falls Sie sich für mehrere LFS-Partitionen entschieden haben (z. B. eine für `/` und eine andere für `/usr`), dann gehen Sie für die restlichen Partitionen gleichermaßen vor:

```
mkdir -pv $LFS
mount -v -t ext3 /dev/<xxx> $LFS
mkdir -v $LFS/usr
mount -v -t ext3 /dev/<yyy> $LFS/usr
```

Natürlich müssen Sie auch hier wieder für *<xxx>* und *<yyy>* die korrekten Bezeichnungen einsetzen.

Die Zugriffsrechte für die neue Partition sollten nicht zu restriktiv sein (wie zum Beispiel mit den Optionen „nosuid“, „nodev“ oder „noatime“). Rufen Sie **mount** ohne Parameter auf, damit Sie sehen, mit welchen Optionen Ihre LFS-Dateisysteme eingehängt wurden. Wenn Optionen wie *nosuid*, *nodev* oder *noatime* aktiviert sind, müssen Sie die Partition erneut einhängen und diese Optionen weglassen.

Wenn Sie eine `swap`-Partition verwenden, stellen Sie bitte sicher, dass diese mittels **swapon** aktiviert ist:

```
/sbin/swapon -v /dev/<zzz>
```

Bitte setzen Sie statt *<xxx>* die Bezeichnung der Swap-Partition ein.

Jetzt haben Sie genügend Platz zum Arbeiten geschaffen und können mit dem Herunterladen der Pakete beginnen.

Kapitel 3. Pakete und Patches

3.1. Einführung

Die folgende Liste enthält alle Pakete, die Sie für ein minimales Linux-System benötigen. Die Versionsnummern sind Versionen, von denen wir *wissen*, dass Sie funktionieren. Wenn Sie noch wenig Erfahrung mit LFS haben sollten Sie lieber keine anderen Versionen probieren. Die Anleitungen und Kommandos könnten evtl. mit neueren Versionen nicht mehr funktionieren. Oft gibt es auch gute Gründe dafür, nicht die allerneueste Version einzusetzen: zum Beispiel bei bekannten Problemen für die es noch keine Lösung gibt.

Wir können nicht für die ständige Verfügbarkeit der Download-Ressourcen garantieren. Falls sich eine Download-Adresse nach Erscheinen des Buches geändert haben sollte, nutzen Sie bitte Google oder eine andere Suchmaschine und suchen nach dem entsprechenden Paket (*http://www.google.com/*). Sollten Sie auch hier erfolglos sein, dann nutzen Sie bitte eine der alternativen Download-Möglichkeiten wie unter *http://www.linuxfromscratch.org/lfs/packages.html* beschrieben.

Sie müssen alle heruntergeladenen Pakete und Patches an einem Ort speichern, auf den Sie während der ganzen Zeit bequemen zugreifen können. Außerdem benötigen Sie einen Arbeitsordner zum Entpacken und Kompilieren der Quellen. Am besten benutzen Sie den Ordner `$LFS/sources` sowohl zum Speichern der Quellen und Patches *als auch* als Arbeitsordner. Damit haben Sie alles Nötige immer auf der LFS-Partition und in allen Arbeitsschritten des Buches verfügbar.

Sie sollten folgendes Kommando als Benutzer `root` auszuführen, bevor Sie mit dem Herunterladen der Pakete beginnen:

```
mkdir -v $LFS/sources
```

Machen Sie den Ordner für jeden beschreibbar und sticky. Der „Sticky“-Modus bewirkt, dass jeweils nur der Besitzer einer Datei diese auch löschen kann, selbst wenn mehrere Benutzer Schreibrechte in dem Ordner haben. Das folgende Kommando schaltet Schreib- und Sticky-Berechtigungen ein:

```
chmod -v a+wt $LFS/sources
```

3.2. Alle Pakete

Bitte laden Sie die folgenden Pakete herunter:

- Autoconf (2.59) - 904 KB:
  Webseite: *http://www.gnu.org/software/autoconf/*
  Download: *http://ftp.gnu.org/gnu/autoconf/autoconf-2.59.tar.bz2*
  MD5-Prüfsumme: `1ee40f7a676b3cfdc0e3f7cd81551b5f`
- Automake (1.9.6) - 748 KB:
  Webseite: *http://www.gnu.org/software/automake/*
  Download: *http://ftp.gnu.org/gnu/automake/automake-1.9.6.tar.bz2*
  MD5-Prüfsumme: `c11b8100bb311492d8220378fd8bf9e0`
- Bash (3.1) - 2,475 KB:
  Webseite: *http://www.gnu.org/software/bash/*
  Download: *http://ftp.gnu.org/gnu/bash/bash-3.1.tar.gz*
  MD5-Prüfsumme: `ef5304c4b22aaa5088972c792ed45d72`
- Bash Dokumentation (3.1) - 2,013 KB:
  Download: *http://ftp.gnu.org/gnu/bash/bash-doc-3.1.tar.gz*
  MD5-Prüfsumme: `a8c517c6a7b21b8b855190399c5935ae`
- Berkeley DB (4.4.20) - 7,767 KB:
  Webseite: *http://dev.sleepycat.com/*
  Download: *http://downloads.sleepycat.com/db-4.4.20.tar.gz*
  MD5-Prüfsumme: `d84dff288a19186b136b0daf7067ade3`
- Binutils (2.16.1) - 12,256 KB:
  Webseite: *http://sources.redhat.com/binutils/*
  Download: *http://ftp.gnu.org/gnu/binutils/binutils-2.16.1.tar.bz2*
  MD5-Prüfsumme: `6a9d529efb285071dad10e1f3d2b2967`
- Bison (2.2) - 1,052 KB:
  Webseite: *http://www.gnu.org/software/bison/*
  Download: *http://ftp.gnu.org/gnu/bison/bison-2.2.tar.bz2*
  MD5-Prüfsumme: `e345a5d021db850f06ce49eba78af027`
- Bzip2 (1.0.3) - 654 KB:
  Webseite: *http://www.bzip.org/*
  Download: *http://www.bzip.org/1.0.3/bzip2-1.0.3.tar.gz*
  MD5-Prüfsumme: `8a716bebecb6e647d2e8a29ea5d8447f`
- Coreutils (5.96) - 4,948 KB:
  Webseite: *http://www.gnu.org/software/coreutils/*
  Download: *http://ftp.gnu.org/gnu/coreutils/coreutils-5.96.tar.bz2*
  MD5-Prüfsumme: `bf55d069d82128fd754a090ce8b5acff`

- DejaGNU (1.4.4) - 1,056 KB:
  Webseite: *http://www.gnu.org/software/dejagnu/*
  Download: *http://ftp.gnu.org/gnu/dejagnu/dejagnu-1.4.4.tar.gz*
  MD5-Prüfsumme: `053f18fd5d00873de365413cab17a666`
- Diffutils (2.8.1) - 762 KB:
  Webseite: *http://www.gnu.org/software/diffutils/*
  Download: *http://ftp.gnu.org/gnu/diffutils/diffutils-2.8.1.tar.gz*
  MD5-Prüfsumme: `71f9c5ae19b60608f6c7f162da86a428`
- E2fsprogs (1.39) - 3,616 KB:
  Webseite: *http://e2fsprogs.sourceforge.net/*
  Download: *http://prdownloads.sourceforge.net/e2fsprogs/e2fsprogs-1.39.tar.gz?download*
  MD5-Prüfsumme: `06f7806782e357797fad1d34b7ced0c6`
- Expect (5.43.0) - 514 KB:
  Webseite: *http://expect.nist.gov/*
  Download: *http://expect.nist.gov/src/expect-5.43.0.tar.gz*
  MD5-Prüfsumme: `43e1dc0e0bc9492cf2e1a6f59f276bc3`
- File (4.17) - 544 KB:
  Download: *ftp://ftp.gw.com/mirrors/pub/unix/file/file-4.17.tar.gz*
  MD5-Prüfsumme: `50919c65e0181423d66bb25d7fe7b0fd`



Anmerkung

Wenn Sie diese Anmerkung lesen ist File (4.17) möglicherweise nicht mehr in dieser Version verfügbar. Der Hauptdownloadserver ist dafür bekannt, alte Versionen zu löschen, sobald neuere verfügbar sind. Bitte nutzen Sie eine der alternativen Download-Adressen wie z. B. *http://www.linuxfromscratch.org/lfs/download.html#ftp*.

- Findutils (4.2.27) - 1,097 KB:
  Webseite: *http://www.gnu.org/software/findutils/*
  Download: *http://ftp.gnu.org/gnu/findutils/findutils-4.2.27.tar.gz*
  MD5-Prüfsumme: `f1e0ddf09f28f8102ff3b90f3b5bc920`
- Flex (2.5.33) - 680 KB:
  Webseite: *http://flex.sourceforge.net*
  Download: *http://prdownloads.sourceforge.net/flex/flex-2.5.33.tar.bz2?download*
  MD5-Prüfsumme: `343374a00b38d9e39d1158b71af37150`
- Gawk (3.1.5) - 1,716 KB:
  Webseite: *http://www.gnu.org/software/gawk/*
  Download: *http://ftp.gnu.org/gnu/gawk/gawk-3.1.5.tar.bz2*
  MD5-Prüfsumme: `5703f72d0eea1d463f735aad8222655f`

- GCC (4.0.3) - 32,208 KB:
  Webseite: *http://gcc.gnu.org/*
  Download: *http://ftp.gnu.org/gnu/gcc/gcc-4.0.3/gcc-4.0.3.tar.bz2*
  MD5-Prüfsumme: `6ff1af12c53cbb3f79b27f2d6a9a3d50`
- Gettext (0.14.5) - 6,940 KB:
  Webseite: *http://www.gnu.org/software/gettext/*
  Download: *http://ftp.gnu.org/gnu/gettext/gettext-0.14.5.tar.gz*
  MD5-Prüfsumme: `e2f6581626a22a0de66dce1d81d00de3`
- Glibc (2.3.6) - 13,687 KB:
  Webseite: *http://www.gnu.org/software/libc/*
  Download: *http://ftp.gnu.org/gnu/glibc/glibc-2.3.6.tar.bz2*
  MD5-Prüfsumme: `bfdce99f82d6dbcb64b7f11c05d6bc96`
- Glibc LibIDN add-on (2.3.6) - 99 KB:
  Download: *http://ftp.gnu.org/gnu/glibc/glibc-libidn-2.3.6.tar.bz2*
  MD5-Prüfsumme: `49dbe06ce830fc73874d6b38bdc5b4db`
- Grep (2.5.1a) - 516 KB:
  Webseite: *http://www.gnu.org/software/grep/*
  Download: *http://ftp.gnu.org/gnu/grep/grep-2.5.1a.tar.bz2*
  MD5-Prüfsumme: `52202fe462770fa6be1bb667bd6cf30c`
- Groff (1.18.1.1) - 2,208 KB:
  Webseite: *http://www.gnu.org/software/groff/*
  Download: *http://ftp.gnu.org/gnu/groff/groff-1.18.1.1.tar.gz*
  MD5-Prüfsumme: `511dbd64b67548c99805f1521f82cc5e`
- GRUB (0.97) - 950 KB:
  Webseite: *http://www.gnu.org/software/grub/*
  Download: *ftp://alpha.gnu.org/gnu/grub/grub-0.97.tar.gz*
  MD5-Prüfsumme: `cd3f3eb54446be6003156158d51f4884`
- Gzip (1.3.5) - 324 KB:
  Webseite: *http://www.gzip.org/*
  Download: *ftp://alpha.gnu.org/gnu/gzip/gzip-1.3.5.tar.gz*
  MD5-Prüfsumme: `3d6c191dfd2bf307014b421c12dc8469`
- Iana-Etc (2.10) - 184 KB:
  Webseite: *http://www.sethwklein.net/projects/iana-etc/*
  Download: *http://www.sethwklein.net/projects/iana-etc/downloads/iana-etc-2.10.tar.bz2*
  MD5-Prüfsumme: `53dea53262b281322143c744ca60ffbb`
- Inetutils (1.4.2) - 1,019 KB:
  Webseite: *http://www.gnu.org/software/inetutils/*
  Download: *http://ftp.gnu.org/gnu/inetutils/inetutils-1.4.2.tar.gz*
  MD5-Prüfsumme: `df0909a586ddac2b7a0d62795eea4206`

- IPRoute2 (2.6.16-060323) - 378 KB:
  Webseite: *http://linux-net.osdl.org/index.php/Iproute2*
  Download: *http://developer.osdl.org/dev/iproute2/download/iproute2-2.6.16-060323.tar.gz*
  MD5-Prüfsumme: `f31d4516b35bbfeaa72c762f5959e97c`
- Kbd (1.12) - 618 KB:
  Download: *http://www.kernel.org/pub/linux/utils/kbd/kbd-1.12.tar.bz2*
  MD5-Prüfsumme: `069d1175b4891343b107a8ac2b4a39f6`
- Less (394) - 286 KB:
  Webseite: *http://www.greenwoodsoftware.com/less/*
  Download: *http://www.greenwoodsoftware.com/less/less-394.tar.gz*
  MD5-Prüfsumme: `a9f072ccefa0d315b325f3e9cdbd4b97`
- LFS-Bootskripte (6.2) - 24 KB:
  Download: *http://www.linuxfromscratch.org/lfs/downloads/6.2/lfs-bootscripts-6.2.tar.bz2*
  MD5-Prüfsumme: `45f9efc6b75c26751ddb74d1ad0276c1`
- Libtool (1.5.22) - 2,856 KB:
  Webseite: *http://www.gnu.org/software/libtool/*
  Download: *http://ftp.gnu.org/gnu/libtool/libtool-1.5.22.tar.gz*
  MD5-Prüfsumme: `8e0ac9797b62ba4dcc8a2fb7936412b0`
- Linux (2.6.16.27) - 39,886 KB:
  Webseite: *http://www.kernel.org/*
  Download: *http://www.kernel.org/pub/linux/kernel/v2.6/linux-2.6.16.27.tar.bz2*
  MD5-Prüfsumme: `ebedfe5376efec483ce12c1629c7a5b1`



Anmerkung

Der Linux-Kernel wird relativ oft aktualisiert; meistens weil neu entdeckte Sicherheitslücken geschlossen werden. Sie sollten die neueste verfügbare Version des Linux-Kernels 2.6.16.x verwenden. Um Kompatibilitätsprobleme mit den Bootskripten zu vermeiden, sollten Sie nicht Version 2.6.17 oder neuer verwenden.

- Linux-Libc-Header (2.6.12.0) - 2,481 KB:
  Download: *http://ep09.pld-linux.org/~mmazur/linux-libc-headers/linux-libc-headers-2.6.12.0.tar.bz2*
  MD5-Prüfsumme: `eae2f562afe224ad50f65a6acfb4252c`
- M4 (1.4.4) - 376 KB:
  Webseite: *http://www.gnu.org/software/m4/*
  Download: *http://ftp.gnu.org/gnu/m4/m4-1.4.4.tar.gz*
  MD5-Prüfsumme: `8d1d64dbecf1494690a0f3ba8db4482a`
- Make (3.80) - 900 KB:
  Webseite: *http://www.gnu.org/software/make/*
  Download: *http://ftp.gnu.org/gnu/make/make-3.80.tar.bz2*
  MD5-Prüfsumme: `0bbd1df101bc0294d440471e50feca71`

- Man-DB (2.4.3) - 798 KB:
  Webseite: *http://www.nongnu.org/man-db/*
  Download: *http://savannah.nongnu.org/download/man-db/man-db-2.4.3.tar.gz*
  MD5-Prüfsumme: `30814a47f209f43b152659ba51fc7937`
- Man-pages (2.34) - 1,760 KB:
  Download: *http://www.kernel.org/pub/linux/docs/manpages/man-pages-2.34.tar.bz2*
  MD5-Prüfsumme: `fb8d9f55fef19ea5ab899437159c9420`
- Mktemp (1.5) - 69 KB:
  Webseite: *http://www.mktemp.org/*
  Download: *ftp://ftp.mktemp.org/pub/mktemp/mktemp-1.5.tar.gz*
  MD5-Prüfsumme: `9a35c59502a228c6ce2be025fc6e3ff2`
- Module-Init-Tools (3.2.2) - 166 KB:
  Webseite: *http://www.kerneltools.org/*
  Download: *http://www.kerneltools.org/pub/downloads/module-init-tools/module-init-tools-3.2.2.tar.bz2*
  MD5-Prüfsumme: `a1ad0a09d3231673f70d631f3f5040e9`
- Ncurses (5.5) - 2,260 KB:
  Webseite: *http://dickey.his.com/ncurses/*
  Download: *ftp://invisible-island.net/ncurses/ncurses-5.5.tar.gz*
  MD5-Prüfsumme: `e73c1ac10b4bfc46db43b2ddfd6244ef`
- Patch (2.5.4) - 183 KB:
  Webseite: *http://www.gnu.org/software/patch/*
  Download: *http://ftp.gnu.org/gnu/patch/patch-2.5.4.tar.gz*
  MD5-Prüfsumme: `ee5ae84d115f051d87fcaaef3b4ae782`
- Perl (5.8.8) - 9,887 KB:
  Webseite: *http://www.perl.com/*
  Download: *http://ftp.funet.fi/pub/CPAN/src/perl-5.8.8.tar.bz2*
  MD5-Prüfsumme: `a377c0c67ab43fd96eeec29ce19e8382`
- Procps (3.2.6) - 273 KB:
  Webseite: *http://procps.sourceforge.net/*
  Download: *http://procps.sourceforge.net/procps-3.2.6.tar.gz*
  MD5-Prüfsumme: `7ce39ea27d7b3da0e8ad74dd41d06783`
- Psmisc (22.2) - 239 KB:
  Webseite: *http://psmisc.sourceforge.net/*
  Download: *http://prdownloads.sourceforge.net/psmisc/psmisc-22.2.tar.gz?download*
  MD5-Prüfsumme: `77737c817a40ef2c160a7194b5b64337`
- Readline (5.1) - 1,983 KB:
  Webseite: *http://cnswww.cns.cwru.edu/php/chet/readline/rltop.html*
  Download: *http://ftp.gnu.org/gnu/readline/readline-5.1.tar.gz*
  MD5-Prüfsumme: `7ee5a692db88b30ca48927a13fd60e46`

- Sed (4.1.5) - 781 KB:
  Webseite: *http://www.gnu.org/software/sed/*
  Download: *http://ftp.gnu.org/gnu/sed/sed-4.1.5.tar.gz*
  MD5-Prüfsumme: `7a1cbbbb3341287308e140bd4834c3ba`
- Shadow (4.0.15) - 1,265 KB:
  Download: *ftp://ftp.pld.org.pl/software/shadow/shadow-4.0.15.tar.bz2*
  MD5-Prüfsumme: `a0452fa989f8ba45023cc5a08136568e`



Anmerkung

Wenn Sie diese Anmerkung lesen ist Shadow (4.0.15) möglicherweise nicht mehr in dieser Version verfügbar. Der Hauptdownloadserver ist dafür bekannt, alte Versionen zu löschen, sobald neuere verfügbar sind. Bitte nutzen Sie eine der alternativen Download-Adressen wie z. B. *http://www.linuxfromscratch.org/lfs/download.html#ftp*.

- Sysklogd (1.4.1) - 80 KB:
  Webseite: *http://www.infodrom.org/projects/sysklogd/*
  Download: *http://www.infodrom.org/projects/sysklogd/download/sysklogd-1.4.1.tar.gz*
  MD5-Prüfsumme: `d214aa40beabf7bdb0c9b3c64432c774`
- Sysvinit (2.86) - 97 KB:
  Download: *ftp://ftp.cistron.nl/pub/people/miquels/sysvinit/sysvinit-2.86.tar.gz*
  MD5-Prüfsumme: `7d5d61c026122ab791ac04c8a84db967`
- Tar (1.15.1) - 1,574 KB:
  Webseite: *http://www.gnu.org/software/tar/*
  Download: *http://ftp.gnu.org/gnu/tar/tar-1.15.1.tar.bz2*
  MD5-Prüfsumme: `57da3c38f8e06589699548a34d5a5d07`
- Tcl (8.4.13) - 3,432 KB:
  Webseite: *http://tcl.sourceforge.net/*
  Download: *http://prdownloads.sourceforge.net/tcl/tcl8.4.13-src.tar.gz?download*
  MD5-Prüfsumme: `c6b655ad5db095ee73227113220c0523`
- Texinfo (4.8) - 1,487 KB:
  Webseite: *http://www.gnu.org/software/texinfo/*
  Download: *http://ftp.gnu.org/gnu/texinfo/texinfo-4.8.tar.bz2*
  MD5-Prüfsumme: `6ba369bbfe4afaa56122e65b3ee3a68c`
- Udev (096) - 190 KB:
  Webseite: *http://www.kernel.org/pub/linux/utils/kernel/hotplug/udev.html*
  Download: *http://www.kernel.org/pub/linux/utils/kernel/hotplug/udev-096.tar.bz2*
  MD5-Prüfsumme: `f4effef7807ce1dc91ab581686ef197b`
- Udev-Einrichtung - 4 KB:
  Download: *http://www.linuxfromscratch.org/lfs/downloads/6.2/udev-config-6.2.tar.bz2*
  MD5-Prüfsumme: `9ff2667ab0f7bfe8182966ef690078a0`

- Util-linux (2.12r) - 1,339 KB:
  Download: *http://www.kernel.org/pub/linux/utils/util-linux/util-linux-2.12r.tar.bz2*
  MD5-Prüfsumme: `af9d9e03038481fbf79ea3ac33f116f9`
- Vim (7.0) - 6,152 KB:
  Webseite: *http://www.vim.org*
  Download: *ftp://ftp.vim.org/pub/vim/unix/vim-7.0.tar.bz2*
  MD5-Prüfsumme: `4ca69757678272f718b1041c810d82d8`
- Vim (7.0) Sprachdateien (optional) - 1,228 KB:
  Webseite: *http://www.vim.org*
  Download: *ftp://ftp.vim.org/pub/vim/extra/vim-7.0-lang.tar.gz*
  MD5-Prüfsumme: `6d43efaff570b5c86e76b833ea0c6a04`
- Zlib (1.2.3) - 485 KB:
  Webseite: *http://www.zlib.net/*
  Download: *http://www.zlib.net/zlib-1.2.3.tar.gz*
  MD5-Prüfsumme: `debc62758716a169df9f62e6ab2bc634`

Gesamtgröße der Pakete: ungefähr 180 MB

3.3. Erforderliche Patches

Zusätzlich brauchen Sie auch einige Patches. Diese beheben z. B. kleine Fehler, die vom jeweiligen Betreuer des Pakets noch nicht behoben wurden, oder beinhalten Modifikationen und Anpassungen an unser LFS. Die folgenden Patches werden zum Erstellen von LFS benötigt:

- Bash Upstream Fixes Patch - 23 KB:
  Download: *http://www.linuxfromscratch.org/patches/lfs/6.2/bash-3.1-fixes-8.patch*
  MD5-Prüfsumme: `bc337045fa4c5839babf0306cc9df6d0`
- Bzip2 Bzgrep Sicherheitslücken-Patch - 1.2 KB:
  Download: *http://www.linuxfromscratch.org/patches/lfs/6.2/bzip2-1.0.3-bzgrep_security-1.patch*
  MD5-Prüfsumme: `4eae50e4fd690498f23d3057dfad7066`
- Bzip2 Dokumentations-Patch - 1.6 KB:
  Download: *http://www.linuxfromscratch.org/patches/lfs/6.2/bzip2-1.0.3-install_docs-1.patch*
  MD5-Prüfsumme: `9e5dfbf4814b71ef986b872c9af84488`
- Coreutils Internationalization Fixes Patch - 101 KB:
  Download: *http://www.linuxfromscratch.org/patches/lfs/6.2/coreutils-5.96-i18n-1.patch*
  MD5-Prüfsumme: `3df2e6fdb1b5a5c13afedd3d3e05600f`
- Coreutils Suppress Uptime, Kill, Su Patch - 13 KB:
  Download:
  http://www.linuxfromscratch.org/patches/lfs/6.2/coreutils-5.96-suppress_uptime_kill_su-1.patch
  MD5-Prüfsumme: `227d41a6d0f13c31375153eae91e913d`
- Coreutils Uname Patch - 4.6 KB:
  Download: *http://www.linuxfromscratch.org/patches/lfs/6.2/coreutils-5.96-uname-1.patch*
  MD5-Prüfsumme: `c05b735710fbd62239588c07084852a0`
- Database (Berkeley) Upstream Fixes Patch - 3.8 KB:
  Download: *http://www.linuxfromscratch.org/patches/lfs/6.2/db-4.4.20-fixes-1.patch*
  MD5-Prüfsumme: `32b28d1d1108dfcd837fe10c4eb0fbad`
- Diffutils Internationalization Fixes Patch - 18 KB:
  Download: *http://www.linuxfromscratch.org/patches/lfs/6.2/diffutils-2.8.1-i18n-1.patch*
  MD5-Prüfsumme: `c8d481223db274a33b121fb8c25af9f7`
- Expect Spawn Patch - 6.8 KB:
  Download: *http://www.linuxfromscratch.org/patches/lfs/6.2/expect-5.43.0-spawn-1.patch*
  MD5-Prüfsumme: `ef6d0d0221c571fb420afb7033b3bbba`
- Gawk Segfault Patch - 1.3 KB:
  Download: *http://www.linuxfromscratch.org/patches/lfs/6.2/gawk-3.1.5-segfault_fix-1.patch*
  MD5-Prüfsumme: `7679530d88bf3eb56c42eb6aba342ddb`
- GCC Specs Patch - 15 KB:
  Download: *http://www.linuxfromscratch.org/patches/lfs/6.2/gcc-4.0.3-specs-1.patch*
  MD5-Prüfsumme: `0aa7d4c6be50c3855fe812f6faabc306`
- Glibc Linux Types Patch - 1.1 KB:
  Download: *http://www.linuxfromscratch.org/patches/lfs/6.2/glibc-2.3.6-linux_types-1.patch*
  MD5-Prüfsumme: `30ea59ae747478aa9315455543b5bb43`

- Glibc Inotify Syscall Functions Patch - 1.4 KB:
  Download: *http://www.linuxfromscratch.org/patches/lfs/6.2/glibc-2.3.6-inotify-1.patch*
  MD5-Prüfsumme: `94f6d26ae50a0fe6285530fdbae90bbf`
- Grep RedHat Fixes Patch - 55 KB:
  Download: *http://www.linuxfromscratch.org/patches/lfs/6.2/grep-2.5.1a-redhat_fixes-2.patch*
  MD5-Prüfsumme: `2c67910be2d0a54714f63ce350e6d8a6`
- Groff Debian Patch - 360 KB:
  Download: *http://www.linuxfromscratch.org/patches/lfs/6.2/groff-1.18.1.1-debian_fixes-1.patch*
  MD5-Prüfsumme: `a47c281afdda457ba4033498f973400d`
- GRUB Disk Geometry Patch - 28 KB:
  Download: *http://www.linuxfromscratch.org/patches/lfs/6.2/grub-0.97-disk_geometry-1.patch*
  MD5-Prüfsumme: `bf1594e82940e25d089feca74c6f1879`
- Gzip Security Patch - 2 KB:
  Download: *http://www.linuxfromscratch.org/patches/lfs/6.2/gzip-1.3.5-security_fixes-1.patch*
  MD5-Prüfsumme: `f107844f01fc49446654ae4a8f8a0728`
- Inetutils GCC-4.x Fix Patch - 1.3 KB:
  Download: *http://www.linuxfromscratch.org/patches/lfs/6.2/inetutils-1.4.2-gcc4_fixes-3.patch*
  MD5-Prüfsumme: `5204fbc503c9fb6a8e353583818db6b9`
- Inetutils No-Server-Man-Pages Patch - 4.1 KB:
  Download:
  http://www.linuxfromscratch.org/patches/lfs/6.2/inetutils-1.4.2-no_server_man_pages-1.patch
  MD5-Prüfsumme: `eb477f532bc6d26e7025fcfc4452511d`
- Kbd Backspace/Delete Fix Patch - 11 KB:
  Download: *http://www.linuxfromscratch.org/patches/lfs/6.2/kbd-1.12-backspace-1.patch*
  MD5-Prüfsumme: `692c88bb76906d99cc20446fadfb6499`
- Kbd GCC-4.x Fix Patch - 1.4 KB:
  Download: *http://www.linuxfromscratch.org/patches/lfs/6.2/kbd-1.12-gcc4_fixes-1.patch*
  MD5-Prüfsumme: `615bc1e381ab646f04d8045751ed1f69`
- Linux-Kernel UTF-8 Composing Patch - 11 KB:
  Download: *http://www.linuxfromscratch.org/patches/lfs/6.2/linux-2.6.16.27-utf8_input-1.patch*
  MD5-Prüfsumme: `d67b53e1e99c782bd28d879e11ee16c3`
- Linux Libc Headers Inotify Patch - 4.7 KB:
  Download: *http://www.linuxfromscratch.org/patches/lfs/6.2/linux-libc-headers-2.6.12.0-inotify-3.patch*
  MD5-Prüfsumme: `8fd71a4bd3344380bd16caf2c430fa9b`
- Mktemp Tempfile Patch - 3.5 KB:
  Download: *http://www.linuxfromscratch.org/patches/lfs/6.2/mktemp-1.5-add_tempfile-3.patch*
  MD5-Prüfsumme: `65d73faabe3f637ad79853b460d30a19`
- Module-init-tools Patch - 1.2 KB:
  Download: *http://www.linuxfromscratch.org/patches/lfs/6.2/module-init-tools-3.2.2-modprobe-1.patch*
  MD5-Prüfsumme: `f1e452fdf3b8d7ef60148125e390c3e8`
- Ncurses Fixes Patch - 8.2 KB:
  Download: *http://www.linuxfromscratch.org/patches/lfs/6.2/ncurses-5.5-fixes-1.patch*

MD5-Prüfsumme: `0e033185008f21578c6e4c7249f92cbb`

- Perl Libc Patch - 1.1 KB:
  Download: *http://www.linuxfromscratch.org/patches/lfs/6.2/perl-5.8.8-libc-2.patch*
  MD5-Prüfsumme: `3bf8aef1fb6eb6110405e699e4141f99`
- Readline Upstream Fixes Patch - 3.8 KB:
  Download: *http://www.linuxfromscratch.org/patches/lfs/6.2/readline-5.1-fixes-3.patch*
  MD5-Prüfsumme: `e30963cd5c6f6a11a23344af36cfa38c`
- Sysklogd 8-Bit Cleanness Patch - 0.9 KB:
  Download: *http://www.linuxfromscratch.org/patches/lfs/6.2/sysklogd-1.4.1-8bit-1.patch*
  MD5-Prüfsumme: `cc0d9c3bd67a6b6357e42807cf06073e`
- Sysklogd Fixes Patch - 27 KB:
  Download: *http://www.linuxfromscratch.org/patches/lfs/6.2/sysklogd-1.4.1-fixes-1.patch*
  MD5-Prüfsumme: `508104f058d1aef26b3bc8059821935f`
- Tar GCC-4.x Fix Patch - 1.2 KB:
  Download: *http://www.linuxfromscratch.org/patches/lfs/6.2/tar-1.15.1-gcc4_fix_tests-1.patch*
  MD5-Prüfsumme: `8e286a1394e6bcf2907f13801770a72a`
- Tar Security Fixes Patch - 3.9 KB:
  Download: *http://www.linuxfromscratch.org/patches/lfs/6.2/tar-1.15.1-security_fixes-1.patch*
  MD5-Prüfsumme: `19876e726d9cec9ce1508e3af74dc22e`
- Tar Sparse Fix Patch - 0.9 KB:
  Download: *http://www.linuxfromscratch.org/patches/lfs/6.2/tar-1.15.1-sparse_fix-1.patch*
  MD5-Prüfsumme: `9e3623f7c88d8766878ecb27c980d86a`
- Texinfo Multibyte Fixes Patch - 1.5 KB:
  Download: *http://www.linuxfromscratch.org/patches/lfs/6.2/texinfo-4.8-multibyte-1.patch*
  MD5-Prüfsumme: `6cb5b760cfdd2dd53a0430eb572a8aaa`
- Texinfo Tempfile Fix Patch - 2.2 KB:
  Download: *http://www.linuxfromscratch.org/patches/lfs/6.2/texinfo-4.8-tempfile_fix-2.patch*
  MD5-Prüfsumme: `559bda136a2ac7777ecb67511227af85`
- Util-linux Cramfs Patch - 2.8 KB:
  Download: *http://www.linuxfromscratch.org/patches/lfs/6.2/util-linux-2.12r-cramfs-1.patch*
  MD5-Prüfsumme: `1c3f40b30e12738eb7b66a35b7374572`
- Vim Upstream Fixes Patch - 42 KB:
  Download: *http://www.linuxfromscratch.org/patches/lfs/6.2/vim-7.0-fixes-7.patch*
  MD5-Prüfsumme: `d274219566702b0bafcb83ab4685bbde`
- Vim Man Directories Patch - 4.2 KB:
  Download: *http://www.linuxfromscratch.org/patches/lfs/6.2/vim-7.0-mandir-1.patch*
  MD5-Prüfsumme: `b6426eb4192faba1e867ddd502323f5b`
- Vim Spellfile Patch - 1.2 KB:
  Download: *http://www.linuxfromscratch.org/patches/lfs/6.2/vim-7.0-spellfile-1.patch*
  MD5-Prüfsumme: `98e59e34cb6e16a8d4671247cebd64ee`

Gesamtgröße der Pakete: ungefähr 775.9 KB

Die LFS-Gemeinschaft hat noch zahlreiche weitere Patches erstellt. Die meisten beheben kleine Probleme oder schalten Funktionen ein, die in der Voreinstellung abgeschaltet sind. Durchstöbern Sie ruhig die Patch-Datenbank unter *http://www.linuxfromscratch.org/patches/* und laden Sie zusätzliche Patche herunter.

Kapitel 4. Abschluss der Vorbereitungen

4.1. Die Variable $LFS

Bei der Arbeit mit dem Buch werden Sie häufig mit der Umgebungsvariable `LFS` zu tun haben. Diese Variable sollte immer definiert sein und den Mountpunkt enthalten, den Sie für die LFS-Partition ausgewählt haben. Überprüfen Sie mit dem folgenden Kommando bitte nochmals, ob `LFS` korrekt gesetzt ist:

```
echo $LFS
```

Die Ausgabe muss dem Pfad zu Ihrer LFS-Partition entsprechen! Wenn Sie unserem Beispiel gefolgt sind lautet der Pfad `/mnt/lfs`. Wenn hier etwas nicht stimmt können Sie die Variable jederzeit neu setzen:

```
export LFS=/mnt/lfs
```

Durch diese Variable haben Sie den Vorteil, dass Sie ein Kommando wie z. B. **mkdir $LFS/tools** genau so eingeben können wie Sie es im Buch lesen. Während die Shell den Befehl verarbeitet, wird sie „$LFS“ durch den echten Wert „/mnt/lfs“ ersetzen.

Wenn Sie Ihre Arbeitsumgebung verlassen haben, müssen Sie anschließend den Inhalt von `$LFS` nochmals prüfen. Das gilt auch, wenn Sie z. B. **su** zu `root` oder einem anderen Benutzer ausführen.

4.2. Erstellen des Ordners $LFS/tools

Alle kompilierten Programme aus Kapitel 5 werden unter `$LFS/tools` installiert. Dadurch werden sie von den Programmen getrennt, die später in Kapitel 6 installiert werden. Die hier kompilierten Programme sind nur übergangsweise Hilfsmittel und sollen nicht Teil des endgültigen LFS-Systems werden. Durch die Installation in einen gesonderten Ordner lassen sie sich später leichter wieder entfernen. Außerdem wird so sichergestellt, dass die Programme nicht versehentlich in Ihrem produktiven Host-System enden (in Kapitel 5 könnte das sehr leicht passieren).

Erstellen Sie den Ordner indem Sie als `root` dieses Kommando ausführen:

```
mkdir -v $LFS/tools
```

Im nächsten Schritt erstellen Sie auf Ihrem *Host-System* einen symbolischen Link nach `/tools`. Er zeigt auf den Ordner, den Sie gerade auf der LFS-Partition erstellt haben. Führen Sie dieses Kommando als `root` aus:

```
ln -sv $LFS/tools /
```



Anmerkung

Das obige Kommando ist in dieser Form korrekt; der Befehl **ln** hat verschiedene Syntax-Varianten — bitte lesen Sie erst **info coreutils ln** und `ln(1)` bevor Sie einen vermeintlichen Fehler berichten.

Dieser symbolische Link ermöglicht uns, die Toolchain so zu kompilieren, dass sie immer `/tools` referenziert. Das hat für uns den Vorteil, dass Compiler, Assembler und Linker sowohl in diesem Kapitel (in dem Sie noch einige Programme vom Host-System benutzen) *als auch* im nächsten Kapitel (wenn Sie in die LFS-Partition „chroot'ed“ haben) funktionieren werden. Das liegt daran, dass die Programme immer den gleichen Pfad benutzen können.

4.3. Hinzufügen des LFS-Benutzers

Als `root` eingeloggt können selbst kleine Fehler ein System beschädigen oder gar zerstören. Daher sollten Sie die Pakete in diesem Kapitel mit Hilfe eines unprivilegierten Benutzers kompilieren. Natürlich können Sie Ihren bisherigen Benutzernamen dazu verwenden, aber das Bereitstellen einer sauberen Arbeitsumgebung ist leichter, wenn Sie dazu den Benutzer `lfs` in der ebenfalls neuen Gruppe `lfs` anlegen und diesen für den ganzen Installationsvorgang benutzen. Bitte führen Sie als *root* dieses Kommando aus, um die neue Gruppe und den Benutzer anzulegen:

```
groupadd lfs
useradd -s /bin/bash -g lfs -m -k /dev/null lfs
```

Die Bedeutung der Kommandozeilen-Parameter:

`-s /bin/bash`
: Dies macht die **bash** zur voreingestellten Shell für den Benutzer `lfs`.

`-g lfs`
: Dieser Parameter macht den neuen Benutzer zum Mitglied der Gruppe `lfs`.

`-m`
: Dadurch wird der Persönliche Ordner für `lfs` gleich mitangelegt.

`-k /dev/null`
: Dieser Parameter verhindert das Kopieren der Dateien aus einem Skeleton-Ordner (Voreinstellung ist `/etc/skel`). Als Quelle für den Skeleton-Ordner wird einfach das Null-Gerät eingestellt.

`lfs`
: Dies ist der Name der erzeugten Gruppe und Benutzer.

Wenn Sie als `root` angemeldet sind und zum Benutzer `lfs` wechseln, benötigen Sie dafür kein Passwort. Wenn Sie sich allerdings als Benutzer `lfs` richtig anmelden möchten, müssen Sie dem Benutzer zuerst ein Passwort zuweisen:

```
passwd lfs
```

Geben Sie `lfs` Vollzugriff auf `$LFS/tools`. Dazu machen Sie *lfs* am besten zum Besitzer des Ordners:

```
chown -v lfs $LFS/tools
```

Wenn Sie, wie vorgeschlagen, einen extra Arbeitsordner eingerichtet haben, dann geben Sie dem Benutzer `lfs` auch dort die Besitzrechte:

```
chown -v lfs $LFS/sources
```

Als nächstes melden Sie sich bitte als `lfs` an. Dazu können Sie eine virtuelle Konsole, den Display-Manager oder das folgende Kommando verwenden:

```
su - lfs
```

Das „`-`“ weist **su** an, eine Login-Shell anstelle einer Nicht-Login-Shell zu starten. Der Unterschied zwischen den beiden Arten wird in `bash(1)`und **info bash** erklärt.

4.4. Vorbereiten der Arbeitsumgebung

Um Ihre Arbeitsumgebung für die weiteren Schritte vorzubereiten erstellen Sie zwei Dateien für die **bash**. Geben Sie als Benutzer `lfs` das folgende Kommando ein, um die neue Datei `.bash_profile` zu erzeugen:

```
cat > ~/.bash_profile << "EOF"
exec env -i HOME=$HOME TERM=$TERM PS1='\u:\w\$ ' /bin/bash
EOF
```

Wenn Sie sich als Benutzer `lfs` anmelden, ist die erste Shell üblicherweise eine *Login*-Shell. Diese liest erst die Datei `/etc/profile` Ihres Host-Systems ein (sie enthält meistens Einstellungen zu Umgebungsvariablen), und danach `.bash_profile`. Das Kommando **exec env -i.../bin/bash** in der zweiten Datei ersetzt die laufende Shell durch eine neue mit einer vollständig leeren Umgebung, mit Ausnahme der Variablen `HOME`, `TERM` und `PS1`. Daduch wird sichergestellt, dass keine ungewollten und potentiell gefährlichen Umgebungsvariablen vom Host-System auf unsere Arbeitsumgebung Einfluss nehmen können. Die hier angewendete Technik mag ein wenig befremdlich wirken, führt aber zu unserem Ziel: einer absolut reinen Arbeitsumgebung.

Die neue Instanz der Shell ist eine *Nicht-Login*-Shell; diese liest weder `/etc/profile` noch `.bash_profile` ein. Stattdessen liest sie die Datei `.bashrc`, erstellen Sie sie nun:

```
cat > ~/.bashrc << "EOF"
set +h
umask 022
LFS=/mnt/lfs
LC_ALL=POSIX
PATH=/tools/bin:/bin:/usr/bin
export LFS LC_ALL PATH
EOF
```

Das Kommando **set +h** schaltet die Hash-Funktion der **bash** ab. Normalerweise ist das sogenannte Hashing der Bash eine nützliche Funktion—**Bash** benutzt eine Hash-Tabelle, um sich die Pfade zu ausführbaren Dateien zu merken und vermeidet auf diese Weise ein ständiges Durchsuchen aller Ordner. Beim Bau von LFS müssen Sie jedoch alle neu installierten Werkzeuge sofort nutzen können. Durch Abschalten der Hash-Funktion wird für „interaktive“ Kommandos (**make**, **patch**, **sed**, **cp** und so weiter) immer die neueste verfügbare Version benutzt.

Das Setzen der Dateierzeugungs-Maske (umask) auf 022 bewirkt, dass neu erzeugte Dateien nur durch ihren Besitzer beschreibbar sind, aber für alle anderen les- und ausführbar (wenn der Systemaufruf `open(2)` die üblichen Datei-Modi benutzt, werden alle neu erzeugten Dateien die Rechte 644 und Ordner die Rechte 755 erhalten).

Die Variable `LFS` sollte natürlich auf den von Ihnen gewählten Mountpunkt der LFS-Partition gesetzt sein.

Die Variable `LC_ALL` beeinflusst die Lokalisierung einiger Programme, so dass deren Ausgaben den Konventionen des entsprechenden Landes folgen. Wenn Ihr Host-System eine ältere Glibc-Version als 2.2.4 verwendet, könnte es Probleme geben, wenn `LC_ALL` nicht auf „POSIX“ oder „C“ gesetzt ist. Durch Setzen von `LC_ALL` auf „POSIX“ oder „C“ (die beiden Werte haben die gleiche Wirkung) sollte es beim Hin- und Herwechseln in der chroot-Umgebung keine Probleme geben.

Durch das Voranstellen von `/tools/bin` an die Umgebungsvariable `PATH` werden alle in Kapitel 5 installierten Programme beim Durchsuchen der Pfade als erstes gefunden und von der Shell sofort benutzt. Zusammen mit dem Abschalten der Hash-Funktion der **Bash** wird so das Risiko minimiert, dass eventuell alte Programme vom Host-System benutzt werden, obwohl schon eine neuere Version aus Kapitel 5 auf dem

System existiert.

Um die Arbeitsumgebung endgültig fertig zu stellen, muss das soeben erzeugte Profil eingelesen werden:

```
source ~/.bash_profile
```

4.5. Informationen zu SBUs

Die meisten Leser möchten gerne vorher wissen, wie lange das Kompilieren und Installieren der Pakete dauert. Linux From Scratch wird aber auf so unterschiedlichen Systemen gebaut, dass es unmöglich ist, echte, auch nur annähernd akkurate Zeiten anzugeben: Das größte Paket (Glibc) braucht auf schnellen Maschinen nicht einmal 20 Minuten, aber auf langsamen Maschinen drei Tage oder mehr. Anstatt Ihnen also Zeiteinheiten zu nennen, haben wir uns für die Standard Binutils Unit entschieden (Abgekürzt: SBU).

Das funktioniert so: Das erste zu kompilierende Paket ist Binutils in Kapitel 5. Die Zeit, die Ihr Computer zum Kompilieren dieses Pakets braucht, entspricht einer „Standard Binutils Unit“ bzw. „SBU“. Alle weiteren Kompilierzeiten werden relativ zu dieser Zeit angegeben.

Nehmen Sie als Beispiel ein Paket mit 4,5 SBU. Wenn das Kompilieren der Binutils 10 Minuten gedauert hat, dann dauert es *ungefähr* 45 Minuten, um das Beispielpaket zu bauen. Glücklicherweise sind die meisten Kompilierzeiten kürzer als die der Binutils.

Grundsätzlich sind SBUs relativ ungenau weil sie auf vielen Faktoren basieren, inklusive der GCC-Version des Host-Systems. Auf Mehrprozessormaschinen können SBUs sogar noch ungenauer sein. SBUs sollen Ihnen eine ungefähre Vorstellung davon geben, wieviel Zeit das Installieren eines Pakets benötigt. Die Angaben können allerdings unter Umständen stark abweichen.

Wenn Sie sich aktuelle Zeitangaben für bestimmte Computerkonfigurationen ansehen möchten, schauen Sie doch mal unter *http://www.linuxfromscratch.org/~sbu/*.

4.6. Über die Testsuites

Die meisten Pakete enthalten auch eine Testsuite. Es ist prinzipiell immer eine gute Idee, eine solche Testsuite für neu kompilierte Programme auch durchlaufen zu lassen. So stellen Sie sicher, dass alles korrekt kompiliert wurde. Wenn eine Testsuite alle ihre Tests erfolgreich durchläuft, können Sie ziemlich sicher sein, dass das Paket so funktioniert, wie es der Entwickler vorgesehen hat. Dennoch ist das natürlich kein Garant für absolute Fehlerfreiheit.

Manche Tests sind wichtiger als andere. So zum Beispiel die Tests der Toolchain-Pakete—GCC, Binutils und Glibc (die C Bibliothek)—sind von höchster Bedeutung, weil diese Pakete eine absolut zentrale Rolle für die Funktion des gesamten Systems spielen. Aber seien Sie gewarnt: die Testsuites von GCC und Glibc brauchen sehr viel Zeit, vor allem auf langsamer Hardware. Dennoch wird dringend empfohlen, sie durchlaufen zu lassen!



Anmerkung

Die Erfahrung hat gezeigt, dass man in Kapitel 5 vom Durchlaufen lassen der Testsuites im Grunde nicht viel gewinnt. Das Host-System hat immer einen gewissen Einfluss auf die Tests in dem Kapitel und das verursacht seltsame und unerklärliche Fehler. Und nicht nur das, die in Kapitel 5 erzeugten Werkzeuge sind ohnehin nur temporär und werden später wieder gelöscht. Daher empfehlen wir Ihnen, die Testsuites in Kapitel 5 *nicht* durchlaufen zu lassen. Die Anleitungen dafür sind dennoch vorhanden, um Testern und Entwicklern eine Hilfe zu sein, aber für alle anderen Anwender sind sie nur optional.

Ein weit verbreitetes Problem beim Durchlaufen der Testsuites von Binutils und GCC sind zu wenig zur Verfügung stehende Pseudo-Terminals (PTYs). Ein typisches Symptom dafür sind ungewöhnlich viele fehlgeschlagene Tests. Das kann verschiedene Ursachen haben. Die häufigste Ursache ist, dass das `devpts`-Dateisystem des Host-Systems nicht funktioniert. Dies wird später in Kapitel 5 ausführlicher behandelt.

Manchmal verursachen Testsuites eines Pakets auch falschen Alarm. Sehen Sie im LFS-Wiki unter *http://www.linuxfromscratch.org/lfs/build-logs/6.2/* nach und prüfen Sie, ob diese Fehler normal sind. Das gilt für alle Tests im gesamten Buch.

Kapitel 5. Erstellen eines temporären Systems

5.1. Einführung

In diesem Kapitel werden Sie ein Minimal-Linux kompilieren und installieren. Das System wird gerade genug Werkzeuge beinhalten, um in Kapitel 6 mit dem Bau des endgültigen LFS beginnen zu können. Wir verzichten hierbei weitestgehend auf jeglichen Komfort.

Das Erstellen des Minimal-Systems erfolgt in zwei Schritten: Zuerst erzeugen Sie eine brandneue, Host-unabhängige Toolchain (Compiler, Assembler, Linker und Bibliotheken und ein paar nützliche Werkzeuge). Mit Hilfe der Toolchain können dann im weiteren Verlauf die essentiellen Werkzeuge kompiliert werden.

Die in diesem Kapitel kompilierten Dateien werden im Ordner `$LFS/tools` installiert und sind damit von den restlichen Dateien des Systems sauber getrennt. Die hier kompilierten Programme sind schließlich nur temporär und sollen nicht mit in unser endgültiges LFS-System einfließen.



Wichtig

Alle Kompilier-Anweisungen setzen voraus, dass Sie die **Bash**-Shell einsetzen. Bevor Sie ein Paket installieren, müssen Sie das jeweilige Tar-Archiv bereits als Benutzer `lfs` entpackt und mit **cd** in den entpackten Ordner gewechselt haben. Danach können Sie die jeweilige Installationsanleitung durcharbeiten.

Einige der Pakete werden vor dem Kompilieren gepatcht, aber nur um ein potentielles Problem zu umgehen. Meist wird ein Patch sowohl in diesem als auch im folgenden Kapitel benötigt, manchmal aber auch nur in einem von beiden. Machen Sie sich keine Gedanken, wenn die Installationsanweisungen für einen Patch zu fehlen scheinen. Außerdem werden Sie manchmal beim Installieren eines Patches Warnungen über *offset* oder *fuzzy* sehen. Diese Warnungen sind nicht wichtig, der Patch wird dennoch sauber installiert.

Beim Kompilieren vieler Pakete werden Sie alle möglichen Compiler-Warnungen auf dem Bildschirm bemerken. Das ist normal und kann einfach ignoriert werden. Es handelt sich eben nur um Warnungen—meistens aufgrund der Verwendung veralteter (aber dennoch korrekter) C- oder C++-Syntax. Die C-Standards haben sich im Laufe der Zeit oft verändert, und einige Pakete benutzen immer noch alte Standards, aber das ist kein wirkliches Problem.



Wichtig

Solange nichts anderes angegeben wird, sollten Sie die Quell- und Kompilierordner jedesmal nach dem Installieren eines Pakets löschen. Dadurch verhindern Sie mögliche Fehlkonfigurationen, falls ein Paket später erneut installiert werden muss.

Bevor Sie fortfahren, stellen Sie bitte mit folgendem Kommando sicher, dass die `LFS`-Umgebungsvariable korrekt gesetzt ist:

```
echo $LFS
```

Die Ausgabe muss den Pfad zum Mountpunkt Ihrer LFS-Partition anzeigen. Wenn Sie unserem Beispiel gefolgt sind, sollte er `/mnt/lfs` lauten.

5.2. Technische Anmerkungen zur Toolchain

Dieser Abschnitt soll Ihnen einige technische Details zum gesamten Kompilier- und Installationsprozess erläutern. Sie müssen nicht alles in diesem Abschnitt sofort verstehen, das Meiste ergibt sich von selbst sobald Sie die ersten Pakete installiert haben. Scheuen Sie sich nicht, zwischendurch noch einmal hierhin zurückzublättern und nachzulesen wenn etwas unklar ist.

In Kapitel 5 soll eine gut funktionierende temporäre Arbeitsumgebung erschaffen werden, in die Sie sich später abkapseln und von wo aus Sie in Kapitel 6 ohne Schwierigkeiten ein sauberes endgültiges LFS-System erstellen können. Sie werden sich so weit wie möglich vom Host-System abschotten und eine in sich geschlossene Toolchain erzeugen. Bitte beachten Sie, dass der gesamte Vorgang dafür ausgelegt ist, die Risiken für neue Leser zu minimieren und gleichzeitig den Lerneffekt zu maximieren.



Wichtig

Bevor Sie fortfahren, sollten Sie den Namen der Plattform kennen, auf der Sie LFS installieren; diesen bezeichnet man oft auch als das *Ziel-Tripplet*. Für die meisten Leser wird das Ziel-Tripplet zum Beispiel *i686-pc-linux-gnu* sein. Sie können Ihr Ziel-Tripplet herauszufinden, indem Sie das Skript **config.guess** auszuführen; es wird mit den Quellen vieler Pakete mitgeliefert. Entpacken Sie die Binutils-Quellen und führen Sie das Skript aus: **`./config.guess`**. Notieren Sie die Ausgabe.

Auch den Namen des *dynamischen Linkers* für Ihre Plattform sollten Sie kennen (manchmal wird der Linker auch als *dynamischer Lader* bezeichnet). Bitte verwechseln Sie den dynamischen Linker nicht mit dem Standard-Linker **ld** aus dem Paket Binutils. Der dynamische Linker kommt mit Glibc und seine Aufgabe ist es, die von einem Programm benötigten gemeinsamen Bibliotheken zu finden und zu laden, das Programm zur Ausführung vorzubereiten und schließlich das Programm selbst auszuführen. Im Regelfall wird der Name des dynamischen Linkers `ld-linux.so.2` sein. Für weniger gängige Systeme könnte der Name auch `ld.so.1` sein und auf neueren 64-Bit-Plattformen könnte er sogar völlig verschieden sein. Sie müssten den Namen Ihres dynamischen Linkers herausfinden können, wenn Sie auf Ihrem Host-System in den Ordner `/lib` schauen. Um wirklich sicher zu gehen, können Sie eine beliebige Binärdatei auf Ihrem Host-System überprüfen: **`readelf -l <Name einer Binärdatei> | grep interpreter`**. Notieren Sie die Ausgabe. Eine Referenz, die alle Plattformen abdeckt, finden Sie in der Datei `shlib-versions` im Basisordner des Glibc-Quellordners.

Hier ein paar technische Anmerkungen zum Kompiliervorgang in Kapitel 5:

- Der Kompiliervorgang ist im Grunde ähnlich wie Cross-Kompilieren. Dabei funktionieren Programme im selben Prefix in Kooperation und benutzen dazu ein wenig GNU-„Magie“.
- Durch vorsichtiges Anpassen des Suchpfades für den Standard-Linker erreichen Sie, dass Programme nur gegen die gewünschten Bibliotheken gelinkt werden.
- Durch vorsichtiges Anpassen von **gcc**'s `specs`-Datei teilen Sie dem Compiler mit, welcher Dynamische Linker verwendet wird.

Als erstes wird Binutils installiert, da sowohl GCC als auch Glibc beim Durchlaufen des **configure**-Skriptes einige Tests zum Assembler und Linker durchführen und auf dem Ergebnis basierend bestimmte Funktionen ein- bzw. ausschalten. Das ist wichtiger als man zunächst denken mag. Ein falsch eingerichteter GCC oder Glibc kann zu Fehlern in der Toolchain führen, die erst am Ende der Installation des LFS-Systems bemerkt werden. Zum Glück weisen Fehlschläge beim Durchlaufen der Testsuites im Regelfall auf solche Probleme

hin, bevor zuviel Zeit vergeudet wird.

Binutils installiert seinen Assembler an zwei Stellen, `/tools/bin` und `/tools/$ZIEL_TRIPPLET/bin`. In Wirklichkeit sind die Programme an der einen Stelle mit denen an der anderen durch einen harten Link verknüpft. Ein wichtiger Aspekt des Linkers ist seine Suchreihenfolge für Bibliotheken. Genaue Informationen erhalten Sie mit **ld** und dem Parameter *--verbose*. Zum Beispiel: **ld --verbose | grep SEARCH** gibt die aktuellen Suchpfade und ihre Reihenfolge aus. Sie können sehen, welche Dateien tatsächlich von **ld** verlinkt werden, indem Sie ein Dummy-Programm kompilieren und den Parameter *--verbose* angeben. Zum Beispiel: **gcc dummy.c -Wl,--verbose 2>&1 | grep succeeded** zeigt, dass alle Dateien beim Linken erfolgreich geöffnet werden konnten.

Das nächste zu installierende Paket ist GCC. Während des Durchlaufs von **configure** sehen Sie zum Beispiel:

```
checking what assembler to use...
        /tools/i686-pc-linux-gnu/bin/as
checking what linker to use... /tools/i686-pc-linux-gnu/bin/ld
```

Das ist aus den oben genannten Gründen wichtig. Hier wird auch deutlich, dass GCC's configure-Skript nicht die `PATH`-Ordner durchsucht, um herauszufinden, welche Werkzeuge verwendet werden sollen. Dennoch werden beim tatsächlichen Ausführen von **gcc** nicht unbedingt die gleichen Suchpfade verwendet. Welchen Standard-Linker **gcc** wirklich verwendet, kann man mittels **gcc -print-prog-name=ld** herausfinden.

Detaillierte Informationen erhält man von **gcc**, indem man den Parameter *-v* beim Kompilieren eines Dummy-Programmes übergibt. **gcc -v dummy.c** zum Beispiel gibt Informationen über den Präprozessor, Komilierungs- und Assemblierungsphasen inklusive **gcc**'s Suchpfaden und der Reihenfolge aus.

Das nächste zu installierende Paket ist Glibc. Die wichtigsten Überlegungen zum Kompilieren von Glibc beschäftigen sich mit dem Compiler, Binutils und den Kernel-Headern. Der Compiler ist normalerweise kein Problem, weil Glibc immer den **gcc** nimmt, der in den `PATH`-Ordnern gefunden wird. Die Binutils und die Kernel-Header können da schon etwas schwieriger sein. Daher gehen wir kein Risiko ein und benutzen die verfügbaren **configure**-Optionen, um die korrekten Entscheidungen zu erzwingen. Nach dem Durchlauf von **./configure** können Sie den Inhalt von `config.make` im Ordner `glibc-build` nach den Details durchsuchen. Sie werden ein paar interessante Dinge finden, wie zum Beispiel *CC="gcc -B/tools/bin/"* zum Kontrollieren der verwendeten Binutils, oder die Parameter *-nostdinc* und *-isystem* zum Kontrollieren des Suchpfades des Compilers. Diese Besonderheiten heben einen wichtigen Aspekt von Glibc hervor—Sie ist kompiliertechnisch gesehen eigenständig und nicht von Voreinstellungen der Toolchain abhängig.

Nach der Installation von Glibc nehmen Sie noch ein paar Anpassungen vor; dadurch stellen Sie sicher, dass Suchen und Verlinken nur innerhalb unseres Prefix `/tools` stattfindet. Sie installieren einen angepassten **ld**, welcher einen fest angegebenen Suchpfad auf `/tools/lib` hat. Dann bearbeiten Sie die `specs`-Datei von **gcc** so, dass sie auf den neuen Dynamischen Linker in `/tools/lib` verweist. Der letzte Schritt ist entscheidend für den gesamten Ablauf. Wie oben bereits angemerkt, wird ein fest eingestellter Pfad zum Dynamischen Linker in jeder ausführbaren ELF-Datei eingebettet. Sie können das überprüfen, indem Sie dieses Kommando ausführen: **readelf -l <Name der ausführbaren Datei> | grep interpreter**. Durch das Anpassen der specs-Datei von gcc stellen wir sicher, dass jedes von nun an kompilierte Programm bis zum Ende des Kapitels unseren neuen Dynamischen Linker in `/tools/lib` benutzt.

Weil unbedingt der neue Linker verwendet werden muss, wird der Specs-Patch auch im zweiten Durchlauf von GCC angewendet. Hierbei darf kein Fehler passieren, denn sonst würden die GCC-Programme selbst den Linker aus `/lib` im Host-System verwenden. Eine saubere Trennung vom Host-System wäre dann nicht mehr gegeben und unser Ziel wäre verfehlt.

Im zweiten Durchlauf der Binutils können Sie den configure-Parameter `--with-lib-path` benutzen, um den Bibliotheksuchpfad von **ld** zu kontrollieren. Von diesem Punkt an ist die Toolchain unabhängig. Die verbleibenden Pakete aus Kapitel 5 kompilieren alle mit der neuen Glibc in `/tools` und alles ist in Ordnung.

Aufgrund ihrer bereits erwähnten eigenständigen Natur ist die Glibc das erste wichtige Paket, das Sie nach dem Eintreten in die chroot-Umgebung in Kapitel 6 installieren. Wenn die Glibc erstmal nach `/usr` installiert ist, werden Sie schnell ein paar Voreinstellungen in der Toolchain ändern und dann schreiten Sie mit dem Erstellen des endgültigen LFS-Systems fort.

5.3. Binutils-2.16.1 - Durchlauf 1

Binutils ist eine Sammlung von Software-Entwicklungswerkzeugen. Dazu gehören zum Beispiel Linker, Assembler und weitere Programme für die Arbeit mit Objektdateien.

Geschätzte Kompilierzeit: 1 SBU
Ungefähr benötigter Festplattenplatz: 189 MB

5.3.1. Installation von Binutils

Es ist wichtig, dass Binutils als erstes Paket kompiliert wird, weil Glibc und GCC verschiedene Tests bezüglich Linker und Assembler durchführen und erst daraufhin bestimmte Funktionen aktivieren.

Die Dokumentation zu Binutils empfiehlt, Binutils außerhalb des Quellordners zu kompilieren:

```
mkdir -v ../binutils-build
cd ../binutils-build
```



Anmerkung

Wenn die im Buch angegebenen SBU-Werte einen Nutzen haben sollen, müssen Sie nun die Zeit messen, die Sie zum Kompilieren von Binutils benötigen. Dies ist mit dem folgenden Kommando relativ einfach: **`time { ./configure ... && make && make install; }`**.

Bereiten Sie Binutils zum Kompilieren vor:

```
../binutils-2.16.1/configure --prefix=/tools --disable-nls
```

Die Bedeutung der configure-Parameter:

`--prefix=/tools`
: Dadurch wird das configure-Skript die Binutils-Programme für die Installation nach `/tools` vorbereiten.

`--disable-nls`
: Deaktiviert die Internationalisierung; i18n wird für die temporären Werkzeuge nicht benötigt.

Fahren Sie mit dem Kompilieren des Pakets fort:

```
make
```

Der Kompiliervorgang ist nun abgeschlossen. Normalerweise würden Sie nun die Testsuite durchlaufen lassen, aber in diesem frühen Stadium ist die Testsuite-Umgebung (Tcl, Expect und DejaGNU) noch nicht verfügbar. Außerdem macht es wenig Sinn, die Tests nun laufen zu lassen, denn die Programme aus dem ersten Durchlauf werden sehr bald durch die aus dem zweiten Durchlauf ersetzt.

Installieren Sie das Paket:

```
make install
```

Bereiten Sie nun den Linker auf die späteren „Anpassungen“ vor:

```
make -C ld clean
make -C ld LIB_PATH=/tools/lib
cp -v ld/ld-new /tools/bin
```

Die Bedeutung der make-Parameter:

-C ld clean
Dies weist das Programm make an, alle kompilierten Dateien im Unterordner `ld` zu löschen.

-C ld LIB_PATH=/tools/lib
Dieser Parameter kompiliert alles im Unterordner `ld` erneut. Die Angabe der Makefile-Variable `LIB_PATH` auf der Kommandozeile überschreibt den Standardwert und zeigt auf den temporären Ordner `tools`. Der Wert dieser Variable gibt den Standard-Bibliotheksuchpfad für den Linker an. Sie werden später in diesem Kapitel sehen, wie diese Vorbereitung zur Anwendung kommt.

Details zu diesem Paket finden Sie in Abschnitt 6.11.2, „Inhalt von Binutils“

5.4. GCC-4.0.3 - Durchlauf 1

Das Paket GCC enthält die GNU-Compiler-Sammlung. Darin sind die C- und C++-Compiler enthalten.

Geschätzte Kompilierzeit: 8.2 SBU
Ungefähr benötigter Festplattenplatz: 514 MB

5.4.1. Installation von GCC

Die Dokumentation zu GCC empfiehlt, GCC außerhalb des Quellordners zu kompilieren:

```
mkdir -v ../gcc-build
cd ../gcc-build
```

Bereiten Sie GCC zum Kompilieren vor:

```
../gcc-4.0.3/configure --prefix=/tools \
    --with-local-prefix=/tools --disable-nls --enable-shared \
    --enable-languages=c
```

Die Bedeutung der configure-Parameter:

--with-local-prefix=/tools
Der Sinn dieses Parameters ist es, `/usr/local/include` aus dem Suchpfad von **gcc** zu entfernen. Dies ist nicht absolut zwingend erforderlich, jedoch sollen mögliche Einflüsse aus dem Host-System vermieden werden, daher ist dieser Parameter hier durchaus empfehlenswert.

--enable-shared
Dieser Parameter ermöglicht das Kompilieren von `libgcc_s.so.1` und `libgcc_eh.a`. Die alleinige Existenz von `libgcc_eh.a` stellt sicher, dass das configure-Skript für Glibc (das nächste zu kompilierende Paket) korrekte Ergebnisse erzielt.

--enable-languages=c
Dieser Parameter stellt sicher, dass nur der C-Compiler erzeugt wird.

Fahren Sie mit dem Kompilieren des Pakets fort:

```
make bootstrap
```

Die Bedeutung des make-Parameters:

bootstrap
Dieses make-Target kompiliert GCC nicht einfach nur, sondern kompiliert gleich mehrmals. GCC benutzt die im ersten Durchlauf erzeugten Programme, um sich damit im zweiten Durchlauf selbst zu kompilieren. Darauf folgt der dritte Kompiliervorgang. Abschließend werden die Ergebnisse des zweiten und dritten Kompiliervorgangs verglichen, um sicherzustellen, dass GCC sich selbst problemlos kompilieren konnte. Das bedeutet normalerweise, dass alles korrekt verlaufen ist.

Der Kompiliervorgang ist nun abgeschlossen. Normalerweise würden Sie nun die Testsuite durchlaufen lassen, aber in diesem frühen Stadium ist die Testsuite-Umgebung (Tcl, Expect und DejaGNU) noch nicht verfügbar. Außerdem macht es wenig Sinn, die Tests nun laufen zu lassen, weil die Programme aus dem ersten Durchlauf sehr bald durch die aus dem zweiten Durchlauf ersetzt werden.

Installieren Sie das Paket:

```
make install
```

Zum Abschluss erstellen Sie noch einen symbolischen Link. Viele Programme rufen das Programm **cc** anstelle von **gcc** auf. Dadurch werden Programme generisch gehalten und sind auf verschiedenen Unix-Systemen lauffähig. Denn nicht jedes System hat den GNU C-Compiler installiert. Der Aufruf von **cc** lässt dem Administrator die Wahl, welchen C-Compiler er installieren möchte, solange ein symbolischer Link auf den echten Compiler verweist:

```
ln -vs gcc /tools/bin/cc
```

Details zu diesem Paket finden Sie in Abschnitt 6.12.2, „Inhalt von GCC“

5.5. Linux-Libc-Header-2.6.12.0

Das Paket Linux-Libc-Header enthält die „bereinigten“ Header-Dateien des Linux-Kernels

Geschätzte Kompilierzeit: weniger als 0.1 SBU
Ungefähr benötigter Festplattenplatz: 27 MB

5.5.1. Installation von Linux-Libc-Header

Über Jahre hinweg war es gängige Praxis, in `/usr/include` die Kernel-Header direkt aus dem Kernel-Archiv zu benutzen. Aber in den letzten Jahren sind die Kernel-Entwickler zu dem Schluss gekommen, dass dies keine gute Praxis ist. Als Konsequenz entstand das Projekt Linux-Libc-Header. Es wurde entworfen, um eine konsistente Programmierschnittstelle (API) zu den Kernel-Headern zu gewährleisten.

Installieren Sie die Header-Dateien:

```
cp -Rv include/asm-i386 /tools/include/asm
cp -Rv include/linux /tools/include
```

Wenn Sie keine i386-Architektur verwenden, passen Sie den Befehl entsprechend an.

Details zu diesem Paket finden Sie in Abschnitt 6.7.2, „Inhalt von Linux-Libc-Header“

5.6. Glibc-2.3.6

Glibc enthält die C-Bibliothek. Sie stellt Systemaufrufe und grundlegende Funktionen zur Verfügung (z. B. das Zuweisen von Speicher, Durchsuchen von Ordnern, Öffnen und Schließen sowie Schreiben von Dateien, Zeichenkettenverarbeitung, Mustererkennung, Arithmetik etc.)

Geschätzte Kompilierzeit: 6 SBU
Ungefähr benötigter Festplattenplatz: 325 MB

5.6.1. Installation von Glibc

Die Dokumentation von Glibc empfiehlt, zum Kompilieren einen gesonderten Ordner zu verwenden:

```
mkdir -v ../glibc-build
cd ../glibc-build
```

Als nächstes bereiten Sie Glibc zum Kompilieren vor:

```
../glibc-2.3.6/configure --prefix=/tools \
    --disable-profile --enable-add-ons \
    --enable-kernel=2.6.0 --with-binutils=/tools/bin \
    --without-gd --with-headers=/tools/include \
    --without-selinux
```

Die Bedeutung der configure-Parameter:

--disable-profile
Dadurch werden die Bibliotheken ohne Profiling-Informationen kompiliert. Lassen Sie diesen Parameter weg, wenn Sie mit den temporären Werkzeugen Profiling betreiben möchten.

--enable-add-ons
Dadurch verwendet Glibc NPTL als die Threading-Bibliothek.

--enable-kernel=2.6.0
Dadurch wird die Glibc mit Unterstützung für Kernel der Serie 2.6.x gebaut.

--with-binutils=/tools/bin
Dieser Parameter wird nicht wirklich benötigt, stellt aber sicher, dass in Hinsicht auf die Binutils-Programme beim Kompilieren von Glibc nichts schiefgehen kann.

--without-gd
Das verhindert das kompilieren des Programmes **memusagestat**, welches immer mit Bibliotheken auf dem Host-System verlinkt (libgd, libpng, libz usw.).

--with-headers=/tools/include
Dadurch wird Glibc mit den gerade in den tools-Ordner installierten Kenerl-Headern kompiliert. Auf diese Weise werden alle Funktionen des Kernels erkannt und die Glibc kann entsprechend darauf optimiert werden.

--without-selinux

Wenn das Host-System SELinux-Funktionen hat (so z. B. Fedora Core 3), so würden die SELinux-Funktionen auch in Glibc einkompiliert. Die LFS-Werkzeuge unterstützen diese Erweiterungen aber nicht, daher wird eine so erzeugte Glibc nicht korrekt funktionieren.

Während dieser Phase sehen Sie möglicherweise eine Warnung:

```
configure: WARNING:
*** These auxiliary programs are missing or
*** incompatible versions: msgfmt
*** some features will be disabled.
*** Check the INSTALL file for required versions.
```

Das fehlende oder inkompatible Programm **msgfmt** ist normalerweise harmlos, aber manchmal kann es zu Fehlern beim Durchlaufen der Testsuite führen. **msgfmt** ist Teil von Gettext, welches auf dem Host-System installiert sein sollte. Wenn **msgfmt** zwar vorhanden, aber vollkommen inkompatibel ist, dann sollten Sie das Paket auf dem Host-System aktualisieren. Oder Sie fahren ohne das Paket fort und schauen, ob die Testsuite auch ohne problemlos durchläuft.

Kompilieren Sie das Paket:

```
make
```

Der Kompiliervorgang ist nun abgeschlossen. Wie bereits erwähnt, wird empfohlen, die Testsuite für das temporäre System in diesem Kapitel nicht durchlaufen zu lassen. Falls Sie die Testsuite dennoch ausführen möchten, verwenden Sie dafür dieses Kommando:

```
make check
```

Eine Information über die kritischen Fehler finden Sie im Abschnitt 6.9, „Glibc-2.3.6“

Die Testsuite von Glibc ist stark von einigen Funktionen Ihres Host-Systems abhängig. Glibc-Fehler in diesem Kapitel sind normalerweise nicht kritisch. Erst in Kapitel 6 wird die endgültige Glibc installiert, dort sollten dann die meisten Tests erfolgreich durchlaufen. Allerdings können selbst in Kapitel 6 noch Fehler auftreten, zum Beispiel beim math-Test.

Wenn ein Fehler auftritt, notieren Sie ihn, dann rufen Sie **make check** erneut auf. Die Testsuite sollte dann dort fortfahren, wo sie unterbrochen wurde. Sie können dieses Stoppen und Starten umgehen, indem Sie **make -k check** aufrufen. Aber stellen Sie in diesem Fall sicher, dass Sie die Ausgaben protokollieren, damit Sie später die Logdatei nach den aufgetretenen Fehlern durchsuchen können.

Auch wenn es nur eine harmlose Meldung ist, die Installationsroutine von Glibc wird sich über die fehlende Datei `/tools/etc/ld.so.conf` beschweren. Beheben Sie diese störende Warnung mit:

```
mkdir -v /tools/etc
touch /tools/etc/ld.so.conf
```

Installieren Sie das Paket:

```
make install
```

Verschiedene Länder und Kulturen haben auch unterschiedliche Konventionen zum Kommunizieren. Darunter sind einfache Konventionen wie zum Beispiel das Format für Datum und Uhrzeit, aber auch sehr komplexe Konventionen, wie zum Beispiel die dort gesprochene Sprache. Die „Internationalisierung“ von GNU-Programmen funktioniert mit Hilfe der sogenannten Locales. Installieren Sie nun die Glibc-Locales.



Anmerkung

Wenn Sie, wie empfohlen, die Testsuite in diesem Kapitel nicht laufen lassen, brauchen Sie auch die Locales nicht zu installieren. Sie werden sie dann im nächsten Kapitel installieren. Um sie dennoch zu installieren, benutzen Sie die Anweisungen aus Abschnitt 6.9, „Glibc-2.3.6“

Details zu diesem Paket finden Sie in Abschnitt 6.9.4, „Inhalt von Glibc“

5.7. Anpassen der Toolchain

Jetzt, nachdem die temporären C-Bibliotheken installiert sind, wollen wir alle im Rest des Kapitels kompilierten Werkzeuge gegen diese Bibliotheken verlinken. Um das zu erreichen, müssen Sie den Linker und die specs-Datei des Compilers anpassen.

Der am Ende des ersten Durchlaufes von Binutils angepasste Linker muss umbenannt werden, da er sonst nicht korrekt gefunden und benutzt wird. Sichern Sie zunächst den Ursprünglichen Linker, dann ersetzen Sie ihn durch den angepassten. Außerdem erzeugen Sie eine Verknüpfung auf das Gegenstück in `/tools/$(gcc - dumpmachine)/bin`

```
mv -v /tools/bin/{ld,ld-old}
mv -v /tools/$(gcc -dumpmachine)/bin/{ld,ld-old}
mv -v /tools/bin/{ld-new,ld}
ln -sv /tools/bin/ld /tools/$(gcc -dumpmachine)/bin/ld
```

Von diesem Punkt an wird alles `ausschließlich` gegen die Bibliotheken in `/tools/lib` verlinkt.

Der nächste Schritt ist nun, GCC auf den neuen dynamischen Linker zu verweisen. Legen Sie dazu GCCs „specs“-Datei an einem Ort ab, den GCC standardmäßig einliest. Dann wird der von GCC verwendete dynamische Linker durch einen einfachen **sed**-Aufruf angepasst:

```
SPECFILE=`dirname $(gcc -print-libgcc-file-name)`/specs &&
gcc -dumpspecs > $SPECFILE &&
sed 's@^/lib/ld-linux.so.2@/tools&@g' $SPECFILE > tempspecfile &&
mv -vf tempspecfile $SPECFILE &&
unset SPECFILE
```

Sie können die specs-Datei auch von Hand ändern: ersetzen Sie einfach jedes Vorkommen von „/lib/ld-linux.so.2“ durch „/tools/lib/ld-linux.so.2“.

Danach sollten Sie die specs-Datei überprüfen und sicherstellen, dass alle gewollten Änderungen durchgeführt wurden.



Wichtig

Wenn Sie auf einer Plattform arbeiten, bei der der Name des dynamischen Linkers nicht `ld-linux.so.2` lautet, müssen Sie natürlich statt „ld-linux.so.2“ den korrekten Namen des Linkers für Ihre Plattform einsetzen. Falls nötig, schauen Sie nochmal im Abschnitt Abschnitt 5.2, „Technische Anmerkungen zur Toolchain,“ nach.

Während dem Installationsvorgang durchsucht GCCs **fixincludes**-Skript Ihr System nach möglicherweise zu reparierenden Header-Dateien (sie könnten z. B. Syntaxfehler enthalten) und kopiert die reparierten Dateien dann in einen privaten Include-Ordner. Dabei kann es vorkommen, dass das Skript einige Header-Dateien von Ihrem Host-System repariert und diese dann in den privaten GCC Include-Ordner kopiert. Weil Sie im Rest dieses Kapitels wirklich nur auf die Header-Dateien von GCC und Glibc angewiesen sind, und diese bereits installiert sind, können alle „reparierten“ Header-Dateien problemlos gelöscht werden. Dadurch verhindern Sie, dass Header-Dateien von Ihrem Host-System Einfluss auf das neue LFS-System nehmen können. Führen Sie bitte das folgende Kommando aus, um die Header-Dateien in GCCs privatem Include-Ordner zu löschen:

```
GCC_INCLUDEDIR=`dirname $(gcc -print-libgcc-file-name)`/include &&
find ${GCC_INCLUDEDIR}/* -maxdepth 0 -xtype d -exec rm -rvf '{}' \; &&
rm -vf `grep -l "DO NOT EDIT THIS FILE" ${GCC_INCLUDEDIR}/*` &&
unset GCC_INCLUDEDIR
```



Achtung

An diesem Punkt ist es unbedingt notwendig, die korrekte Funktion der Toolchain (Kompilieren und Linken) zu überprüfen. Darum führen Sie nun einen kleinen „Gesundheitscheck“ durch:

```
echo 'main(){}' > dummy.c
cc dummy.c
readelf -l a.out | grep ': /tools'
```

Wenn alles korrekt funktioniert, sollten keine Fehler auftreten und die Ausgabe des letzten Kommandos sieht so oder so ähnlich aus:

```
[Requesting program interpreter:
    /tools/lib/ld-linux.so.2]
```

Achten Sie besonders darauf, dass `/tools/lib` als Prefix zu Ihrem dynamischen Linker angegeben ist.

Wenn Sie keine oder eine andere als die obige Ausgabe erhalten haben, ist etwas schiefgelaufen. Sie müssen alle Ihre Schritte noch einmal überprüfen und den Fehler finden und korrigieren. Fahren Sie nicht fort, bevor Sie den Fehler nicht beseitigt haben. Als erstes führen Sie nochmals den Gesundheitscheck durch und benutzen **gcc** anstelle von **cc**. Wenn das funktioniert, fehlt der Link von `/tools/bin/cc`. Gehen Sie zurück zu Abschnitt 5.4, „GCC-4.0.3 - Durchlauf 1“ und reparieren Sie den symbolischen Link. Als zweites stellen Sie bitte sicher, dass Ihre Umgebungsvariable `PATH` richtig gesetzt ist. Sie können die Variable mit dem Kommando **echo $PATH** anzeigen lassen; prüfen Sie, dass `/tools/bin` am Anfang der Liste steht. Wenn die `PATH` Variable falsch gesetzt ist, sind Sie möglicherweise nicht als `lfs` eingeloggt oder in Abschnitt 4.4, „Vorbereiten der Arbeitsumgebung“ ist etwas schiefgelaufen. Vielleicht hat auch beim Anpassen der specs-Datei etwas nicht richtig funktioniert. In diesem Fall wiederholen Sie die Anpassung.

Wenn Sie mit dem Ergebnis zufrieden sind, räumen Sie auf:

```
rm -v dummy.c a.out
```



Anmerkung

Das Kompilieren von TCL im nächsten Abschnitt ist gleichzeitig auch ein zusätzlicher Test, ob die Toolchain korrekt erstellt wurde. Falls TCL nicht kompilierbar ist, weist das auf einen Fehler mit Binutils, GCC oder Glibc hin, nicht aber auf einen Fehler in TCL.

5.8. Tcl-8.4.13

Das Tcl Paket enthält die Tool Command Language.

Geschätzte Kompilierzeit: 0.3 SBU
Ungefähr benötigter Festplattenplatz: 24 MB

5.8.1. Installation von Tcl

Dieses und die nächsten beiden Pakete werden nur installiert, damit Sie die Testsuites von GCC und Binutils laufen lassen können. Drei Pakete nur zu Testzwecken zu installieren könnte etwas übertrieben erscheinen, aber es ist wirklich sehr wichtig zu wissen, dass unsere grundlegensten Programme und Werkzeuge richtig funktionieren. Selbst wenn wir die Testsuites in diesem Kapitel nicht ausführen (wie empfohlen), werden diese Pakete doch zumindest für die Tests im nächsten Kapitel 6 benötigt.

Bereiten Sie Tcl zum Kompilieren vor:

```
cd unix
./configure --prefix=/tools
```

Kompilieren Sie das Paket:

```
make
```

Wenn Sie die Testsuite ausführen möchten, führen Sie **`TZ=UTC make test`** aus. Es ist jedoch bekannt, dass die Testsuite von Tcl unter bestimmten Bedingungen fehlschlägt. Daher sind Fehler in der Testsuite nicht überraschend; wir betrachten diese Fehler nicht als kritisch. Der Parameter *`TZ=UTC`* setzt die Zeitzone für die Dauer des Durchlaufs der Testsuite auf Coordinated Universal Time (UTC), auch als Greenwhich Mean Time (GMT) bekannt. Dadurch werden zeitbezogene Tests korrekt ausgewertet. Mehr Informationen zu der Umgebungsvariable `TZ` finden Sie später in Kapitel 7.

Installieren Sie das Paket:

```
make install
```

Installieren Sie die Tcl Header-Dateien. Das nächste Paket (Expect) benötigt Sie zum Kompilieren.

```
make install-private-headers
```

Erstellen Sie einen nötigen symbolischen Link:

```
ln -sv tclsh8.4 /tools/bin/tclsh
```

5.8.2. Inhalt von Tcl

Installierte Programme: tclsh (Link auf tclsh8.4) uand tclsh8.4
Installierte Bibliothek: libtcl8.4.so

Kurze Beschreibungen

tclsh8.4	Die Tcl Kommando-Shell.
tclsh	Ein Link auf **tclsh8.4**.
`libtcl8.4.so`	Die Tcl-Bibliothek

5.9. Expect-5.43.0

Das Paket Expect führt vorprogrammierte Dialoge mit anderen interaktiven Programmen aus.

Geschätzte Kompilierzeit: 0.1 SBU
Ungefähr benötigter Festplattenplatz: 4 MB

5.9.1. Installation von Expect

Spielen Sie erst einen Patch ein; er behebt einen Fehler, der ansonsten Fehlalarme beim Durchlaufen von GCCs Testsuite verursachen könnte:

```
patch -Np1 -i ../expect-5.43.0-spawn-1.patch
```

Bereiten Sie Expect nun zum Kompilieren vor:

```
./configure --prefix=/tools --with-tcl=/tools/lib \
  --with-tclinclude=/tools/include --with-x=no
```

Die Bedeutung der configure-Parameter:

--with-tcl=/tools/lib
: So stellen Sie sicher, dass das configure-Skript die Tcl-Installation in Ihrem temporären Ordner findet. Es sollte keine möglicherweise auf dem Host-System installierte Version gefunden werden.

--with-tclinclude=/tools/include
: Durch diesen Parameter wird Expect mitgeteilt, wo die Header von Tcl zu finden sind. Dadurch wird ein Fehlschlagen von **configure** vermieden, falls es die Tcl-Header nicht automatisch auffinden kann.

--with-x=no
: Dies teilt dem configure-Skript mit, dass es nicht nach Tk (der grafischen Oberfläche zu Tcl) oder den X Window-Bibliotheken suchen soll; beide könnten eventuell auf dem Host-System existieren, fehlen aber in der temporären Arbeitsumgebung.

Kompilieren Sie das Paket:

```
make
```

Wenn Sie die Testsuite durchlaufen lassen möchten, führen Sie **make test** aus. Es ist jedoch bekannt, dass die Testsuite in diesem Kapitel Probleme macht, die noch nicht ganz nachvollzogen wurden. Es ist daher nicht überraschend, wenn die Testsuite Fehler meldet, diese werden jedoch nicht als kritisch betrachtet.

Installieren Sie das Paket:

```
make SCRIPTS="" install
```

Die Bedeutung des make-Parameters:

SCRIPTS=""
: Dies verhindert die Installation der mitgelieferten Expect-Skripte, sie werden hier nicht gebraucht.

5.9.2. Inhalt von Expect

Installiertes Programm: expect
Installierte Bibliothek: libexpect-5.43.a

Kurze Beschreibungen

expect	Expect „Spricht“ mit anderen interaktiven Programmen. Es verwendet dafür ein anpassbares Skript.
`libexpect-5.43.a`	Enthält Funktionen, mit denen man Expect als TCL-Erweiterung oder direkt aus C/C++ (ohne TCL) nutzen kann

5.10. DejaGNU-1.4.4

Das Paket DejaGNU enthält ein Grundgerüst zum Testen anderer Programme.

Geschätzte Kompilierzeit: weniger als 0.1 SBU
Ungefähr benötigter Festplattenplatz: 6.2 MB

5.10.1. Installation von DejaGNU

Bereiten Sie DejaGNU zum Kompilieren vor:

```
./configure --prefix=/tools
```

Kompilieren und installieren Sie das Paket:

```
make install
```

Um das Ergebnis zu prüfen, führen Sie **make check** aus.

5.10.2. Inhalt von DejaGNU

Installiertes Programm: runtest

Kurze Beschreibungen

runtest Das Wrapper-Skript, das die korrekte **expect**-Shell findet und DejaGNU ausführt.

5.11. GCC-4.0.3 - Durchlauf 2

Das Paket GCC enthält die GNU-Compiler-Sammlung. Darin sind die C- und C++-Compiler enthalten.

Geschätzte Kompilierzeit: 4.2 SBU
Ungefähr benötigter Festplattenplatz: 443 MB

5.11.1. Neuinstallation von GCC

Die Hilfsmittel zum Testen von GCC und Binutils sind nun installiert (Tcl, Expect und DejaGNU). Sie können GCC und Binutils nun erneut installieren, gegen die neue Glibc verlinken und testen. Eines muss noch beachtet werden: Die Testsuites sind stark von funktionierenden Pseudo-Terminals (PTYs) abhängig. Diese werden vom Host-System bereitgestellt. Heutzutage werden PTYs meist über das Dateisystem `devpts` implementiert. Ob Ihr Host-System korrekt eingerichtet ist, können Sie mit einem einfachen Test feststellen:

```
expect -c "spawn ls"
```

Das Ergebnis könnte so aussehen:

```
The system has no more ptys.
Ask your system administrator to create more.
```

Wenn Sie die obige Meldung sehen, ist Ihr Host-System nicht korrekt für PTYs eingerichtet. Solange Sie dieses Problem nicht behoben haben, brauchen Sie die Testsuites von GCC und Binutils gar nicht erst durchlaufen lassen. Wenn Sie mehr Informationen zum Einrichten von PTYs brauchen, schauen Sie am besten in die LFS-FAQ unter *http://www.linuxfromscratch.org//lfs/faq.html#no-ptys*.

In Abschnitt 5.7, „Anpassen der Toolchain“ wurde bereits erklärt, dass GCC unter normalen Umständen sein **fixincludes**-Skript laufen lässt, um defekte Header-Dateien aufzufinden und zu reparieren. Da an diesem Punkt GCC-4.0.3 und Glibc-2.3.6 bereits installiert sind und deren Header-Dateien definitiv nicht repariert werden müssen, wird das **fixincludes**-Skript eigentlich nicht benötigt. Wie bereits erwähnt, könnte es sogar den negativen Nebeneffekt haben, Header-Dateien vom Host-System in das LFS-System einzuschleusen. Mit dem folgenden Kommando können Sie das Ausführen des **fixincludes**-Skriptes verhindern:

```
cp -v gcc/Makefile.in{,.orig} &&
sed 's@\./fixinc\.sh@-c true@' gcc/Makefile.in.orig > gcc/Makefile.in
```

Im Bootstrap-Durchlauf aus Abschnitt 5.4, „GCC-4.0.3 - Durchlauf 1“ wurde zum Kompilieren von GCC der Compiler-Parameter `-fomit-frame-pointer` verwendet. Der Nicht-Bootstrap-Durchlauf verwendet diesen Parameter jedoch standardmäßig nicht. Um die Kompilier-Durchläufe von GCC konsistent zu halten, sollten Sie den Parameter für diesen Durchlauf mit dem folgenden **sed**-Kommando einschalten.

```
cp -v gcc/Makefile.in{,.tmp} &&
sed 's/^XCFLAGS =$/& -fomit-frame-pointer/' gcc/Makefile.in.tmp \
  > gcc/Makefile.in
```

Wenden Sie nun den folgenden Patch an. Dadurch wird der Pfad zu GCCs voreingestelltem dynamischen Linker festgelegt (üblicherweise `ld-linux.so.2`):

```
patch -Np1 -i ../gcc-4.0.3-specs-1.patch
```

Desweiteren entfernt der obige Patch `/usr/include` aus dem Include-Suchpfad von GCC. Das Patchen an dieser Stelle statt des nachträglichen Anpassens der specs-Datei stellt sicher, dass beim Kompilieren von GCC der neue dynamische Linker verwendet wird. Dies bedeutet, dass alle Binärdateien beim Kompiliervorgang gegen die neue Glibc gelinkt werden.



Wichtig

Diese Patches sind zwingende Voraussetzung für einen erfolgreichen Gesamtdurchlauf. Vergessen Sie nicht, sie zu installieren!

Erstellen Sie erneut einen eigenen Ordner zum Kompilieren:

```
mkdir -v ../gcc-build
cd ../gcc-build
```

Denken Sie daran, vor dem Kompilieren von GCC alle Umgebungsvariablen zurückzusetzen, die die Standard-Optimierungen überschreiben würden.

Bereiten Sie GCC zum Kompilieren vor:

```
../gcc-4.0.3/configure --prefix=/tools \
    --with-local-prefix=/tools --enable-clocale=gnu \
    --enable-shared --enable-threads=posix \
    --enable-__cxa_atexit --enable-languages=c,c++ \
    --disable-libstdcxx-pch
```

Die Bedeutung der neuen Parameter zu configure:

`--enable-clocale=gnu`
: Dieser Parameter stellt sicher, dass unter allen Umständen das korrekte locale-Modell für die C++ Bibliotheken ausgewählt wird. Falls das configure-Skript *de_DE* Locales findet, wird es das korrekte Modell gnu wählen. Falls aber *de_DE* nicht installiert ist, besteht das Risiko, dass aufgrund des fälschlicherweise ausgewählten Modells generic ABI-inkompatible C++-Bibliotheken erstellt werden.

`--enable-threads=posix`
: Das schaltet die Behandlung von C++-Exceptions für Code mit Threads ein.

`--enable-__cxa_atexit`
: Dieser Parameter ermöglicht die Verwendung von `__cxa_atexit` anstelle von `atexit`, um C++-Destruktoren für lokale Statics und globale Objekte zu registrieren. Außerdem ist die Option für eine vollständig standardkonforme Behandlung von Destruktoren erforderlich. Das beeinflusst auch die C++ ABI; das Ergebnis sind gemeinsame C++-Bibliotheken und C++-Programme die interoperabel mit anderen Linux-Distributionen sind.

`--enable-languages=c,c++`
: Dieser Parameter stellt sicher, dass sowohl der C- als auch der C++-Compiler erzeugt werden.

`--disable-libstdcxx-pch`
: Verhindert das Erzeugen der vorkompilierten Header-Dateien (PCH, pre-compiled header) für `libstdc++`. Diese Funktion verbraucht viel Platz und wir benötigen sie nicht.

Kompilieren Sie das Paket:

```
make
```

Diesmal müssen Sie nicht das *bootstrap*-Target verwenden, weil Sie bereits einen Compiler benutzen, der aus exakt den gleichen Quellen gebaut wurde.

Der Kompiliervorgang ist nun abgeschlossen. Wie bereits erwähnt, empfehlen wir, die Testsuite für das temporäre System in diesem Kapitel nicht durchlaufen zu lassen. Falls Sie die Testsuite dennoch laufen lassen möchten, führen Sie dieses Kommando aus:

```
make -k check
```

Der Parameter *-k* lässt die Testsuite bis zum Ende durchlaufen, selbst wenn Fehler auftreten sollten. Die Testsuite von GCC ist sehr umfangreich und es ist beinahe sicher, dass Fehler auftreten.

Eine Information über die kritischen Fehler finden Sie im Abschnitt 6.12, „GCC-4.0.3“

Installieren Sie das Paket:

```
make install
```



Achtung

An diesem Punkt ist es unbedingt notwendig, die korrekte Funktion der Toolchain (Kompilieren und Linken) zu überprüfen. Darum führen Sie nun einen kleinen „Gesundheitscheck“ durch:

```
echo 'main(){}' > dummy.c
cc dummy.c
readelf -l a.out | grep ': /tools'
```

Wenn alles korrekt funktioniert, sollten keine Fehler auftreten und die Ausgabe des letzten Kommandos sieht so oder so ähnlich aus:

```
[Requesting program interpreter:
    /tools/lib/ld-linux.so.2]
```

Achten Sie besonders darauf, dass `/tools/lib` als Prefix zu Ihrem dynamischen Linker angegeben ist.

Wenn Sie keine oder eine andere als die obige Ausgabe erhalten haben, ist etwas schiefgelaufen. Sie müssen alle Ihre Schritte noch einmal überprüfen und den Fehler finden und korrigieren. Fahren Sie nicht fort, bevor Sie den Fehler nicht beseitigt haben. Als erstes führen Sie nochmals den Gesundheitscheck durch und benutzen **gcc** anstelle von **cc**. Wenn das funktioniert, fehlt der Link von `/tools/bin/cc`. Gehen Sie zurück zu Abschnitt 5.4, „GCC-4.0.3 - Durchlauf 1“ und reparieren Sie den symbolischen Link. Als zweites stellen Sie bitte sicher, dass Ihre Umgebungsvariable PATH richtig gesetzt ist. Sie können die Variable mit dem Kommando **echo $PATH** anzeigen lassen; prüfen Sie, dass `/tools/bin` am Anfang der Liste steht. Wenn die PATH Variable falsch gesetzt ist, sind Sie möglicherweise nicht als `lfs` eingeloggt oder in Abschnitt 4.4, „Vorbereiten der Arbeitsumgebung“ ist etwas schiefgelaufen. Vielleicht hat auch beim Anpassen der specs-Datei etwas nicht richtig funktioniert. In diesem Fall wiederholen Sie die Anpassung.

Wenn Sie mit dem Ergebnis zufrieden sind, räumen Sie auf:

```
rm -v dummy.c a.out
```

Details zu diesem Paket finden Sie in Abschnitt 6.12.2, „Inhalt von GCC“

5.12. Binutils-2.16.1 - Durchlauf 2

Binutils ist eine Sammlung von Software-Entwicklungswerkzeugen. Dazu gehören zum Beispiel Linker, Assembler und weitere Programme für die Arbeit mit Objektdateien.

Geschätzte Kompilierzeit: 1.1 SBU
Ungefähr benötigter Festplattenplatz: 154 MB

5.12.1. Neuinstallation von Binutils

Erstellen Sie erneut einen eigenen Ordner zum Kompilieren:

```
mkdir -v ../binutils-build
cd ../binutils-build
```

Bereiten Sie Binutils zum Kompilieren vor:

```
../binutils-2.16.1/configure --prefix=/tools \
    --disable-nls --with-lib-path=/tools/lib
```

Die Bedeutung der neuen Parameter zu configure:

--with-lib-path=/tools/lib
: Dies teilt dem configure-Skript mit, den Standard Bibliotheksuchpfad des Linkers als `/tools/lib` vorzugeben. Wir möchten im Standard Bibliotheksuchpfad keine Ordner unseres Host-Systems haben, daher geben Sie den gewünschten Pfad vor.

Kompilieren Sie das Paket:

```
make
```

Der Kompiliervorgang ist nun abgeschlossen. Wie bereits erwähnt, wird empfohlen, die Testsuite für das temporäre System in diesem Kapitel nicht durchlaufen zu lassen. Falls Sie die Testsuite dennoch laufen lassen möchten, führen Sie dieses Kommando aus:

```
make check
```

Installieren Sie das Paket:

```
make install
```

Nun bereiten Sie Binutils auf das erneute Anpassen der Toolchain im nächsten Kapitel vor:

```
make -C ld clean
make -C ld LIB_PATH=/usr/lib:/lib
cp -v ld/ld-new /tools/bin
```

Details zu diesem Paket finden Sie in Abschnitt 6.11.2, „Inhalt von Binutils“

5.13. Ncurses-5.5

Das Paket Ncurses enthält Bibliotheken für den Terminal-unabhängigen Zugriff auf Textbildschirme.

Geschätzte Kompilierzeit: 0.7 SBU
Ungefähr benötigter Festplattenplatz: 30 MB

5.13.1. Installation von Ncurses

Bereiten Sie Ncurses zum Kompilieren vor:

```
./configure --prefix=/tools --with-shared \
    --without-debug --without-ada --enable-overwrite
```

Die Bedeutung der configure-Parameter:

--without-ada
Dies stellt sicher, dass Ncurses ohne Unterstützung für Ada-Compiler erzeugt wird. Auf dem Host-System könnte Unterstützung für Ada installiert sein. Sie wäre dann aber später in der **chroot**-Umgebung nicht mehr verfügbar.

--enable-overwrite
Dadurch werden die Header-Dateien von Ncurses in `/tools/include` anstelle von `/tools/include/ncurses` installiert. Das stellt sicher, dass andere Pakete die Header-Dateien problemlos finden können.

Kompilieren Sie das Paket:

```
make
```

Dieses Paket enthält keine Testsuite.

Installieren Sie das Paket:

```
make install
```

Details zu diesem Paket finden Sie in Abschnitt 6.18.2, „Inhalt von Ncurses“

5.14. Bash-3.1

Das Paket Bash enthält die Bourne-Again-SHell.

Geschätzte Kompilierzeit: 0.4 SBU
Ungefähr benötigter Festplattenplatz: 22 MB

5.14.1. Installation von Bash

Die Upstream-Entwickler haben seit der ersten Veröffentlichung von Bash-3.1 viele Fehler behoben. Wenden Sie diese Fehlerkorrekturen nun an:

```
patch -Np1 -i ../bash-3.1-fixes-8.patch
```

Bereiten Sie Bash zum Kompilieren vor:

```
./configure --prefix=/tools --without-bash-malloc
```

Die Bedeutung des configure-Parameters:

--without-bash-malloc
Dieser Parameter schaltet Bash's memory allocation (`malloc`) Funktion ab; sie ist dafür bekannt, Speicherzugriffsfehler zu verursachen. Durch das Abschalten der Funktion, wird Bash die stabilere `malloc`-Funktion von Glibc benutzen.

Kompilieren Sie das Paket:

```
make
```

Zum Testen der Ergebnisse führen Sie dieses Kommando aus: **make tests**.

Installieren Sie das Paket:

```
make install
```

Und erstellen Sie einen Link für die Programme, die **sh** als Shell benutzen:

```
ln -vs bash /tools/bin/sh
```

Details zu diesem Paket finden Sie in Abschnitt 6.27.2, „Inhalt von Bash“

5.15. Bzip2-1.0.3

Das Paket Bzip2 enthält Programme zum Komprimieren und Dekomprimieren von Dateien. **Bzip2** erreicht vor allem bei Textdateien eine wesentlich bessere Kompressionsrate als das traditionelle **gzip**.

Geschätzte Kompilierzeit: weniger als 0.1 SBU
Ungefähr benötigter Festplattenplatz: 4.2 MB

5.15.1. Installation von Bzip2

Das Paket Bzip2 enthält kein **configure**-Skript. Kompilieren Sie es einfach:

```
make
```

Installieren Sie das Paket:

```
make PREFIX=/tools install
```

Details zu diesem Paket finden Sie in Abschnitt 6.28.2, „Inhalt von Bzip2“

5.16. Coreutils-5.96

Das Paket Coreutils enthält viele Shell-Werkzeuge zum Einstellen der grundlegenden Systemeigenschaften.

Geschätzte Kompilierzeit: 0.6 SBU
Ungefähr benötigter Festplattenplatz: 56.1 MB

5.16.1. Installation von Coreutils

Bereiten Sie Coreutils zum Kompilieren vor:

```
./configure --prefix=/tools
```

Kompilieren Sie das Paket:

```
make
```

Wenn Sie die Testsuite durchlaufen lassen möchten, führen Sie dieses Kommando aus: **make RUN_EXPENSIVE_TESTS=yes check**. Der Parameter *RUN_EXPENSIVE_TESTS=yes* teilt der Testsuite mit, noch zusätzliche Tests zu durchlaufen, die auf einigen Plattformen sehr zeitintensiv sein können. Normalerweise ist das unter Linux aber kein Problem.

Installieren Sie das Paket:

```
make install
```

Details zu diesem Paket finden Sie in Abschnitt 6.14.2, „Inhalt von Coreutils“

5.17. Diffutils-2.8.1

Die Programme dieses Pakets können Unterschiede zwischen Dateien oder Ordnern anzeigen.

Geschätzte Kompilierzeit: 0.1 SBU
Ungefähr benötigter Festplattenplatz: 6.2 MB

5.17.1. Installation von Diffutils

Bereiten Sie Diffutils zum Kompilieren vor:

```
./configure --prefix=/tools
```

Kompilieren Sie das Paket:

```
make
```

Dieses Paket enthält keine Testsuite.

Installieren Sie das Paket:

```
make install
```

Details zu diesem Paket finden Sie in Abschnitt 6.29.2, „Inhalt von Diffutils“

5.18. Findutils-4.2.27

Das Paket Findutils enthält Programme zum Auffinden von Dateien durch rekursive Suche in einer Ordnerstruktur oder über den Zugriff auf eine Datenbank. Die Suche über eine Datenbank ist normalerweise schneller, aber es besteht natürlich die Gefahr, dass die Datenbank zum Zeitpunkt der Suche veraltet ist.

Geschätzte Kompilierzeit: 0.2 SBU
Ungefähr benötigter Festplattenplatz: 12 MB

5.18.1. Installation von Findutils

Bereiten Sie Findutils zum Kompilieren vor:

```
./configure --prefix=/tools
```

Kompilieren Sie das Paket:

```
make
```

Um das Ergebnis zu prüfen, führen Sie **`make check`** aus.

Installieren Sie das Paket:

```
make install
```

Details zu diesem Paket finden Sie in Abschnitt 6.32.2, „Inhalt von Findutils“

5.19. Gawk-3.1.5

Gawk ist eine Implementierung von awk und wird zur Textmanipulation verwendet.

Geschätzte Kompilierzeit: 0.2 SBU
Ungefähr benötigter Festplattenplatz: 18.2 MB

5.19.1. Installation von Gawk

Bereiten Sie Gawk zum Kompilieren vor:

```
./configure --prefix=/tools
```

Aufgrund eines Fehlers im **configure**-Skript erkennt Gawk einige Funktionen von Glibc's locale-Unterstützung nicht richtig. Das führt z. B. zu Fehlern in der Testsuite von Gettext. Sie können das Problem umgehen, indem Sie die fehlenden Makro-Definitionen in der Datei `config.h` hinzufügen:

```
cat >>config.h <<"EOF"
#define HAVE_LANGINFO_CODESET 1
#define HAVE_LC_MESSAGES 1
EOF
```

Kompilieren Sie das Paket:

```
make
```

Um das Ergebnis zu prüfen, führen Sie **`make check`** aus.

Installieren Sie das Paket:

```
make install
```

Details zu diesem Paket finden Sie in Abschnitt 6.35.2, „Inhalt von Gawk“

5.20. Gettext-0.14.5

Gettext wird zur Übersetzung und Lokalisierung verwendet. Programme können mit Unterstützung für NLS (Native Language Support, Unterstützung für die lokale Sprache) kompiliert werden. Dadurch können Texte und Meldungen in der Sprache des Anwenders ausgegeben werden.

Geschätzte Kompilierzeit: 0.4 SBU
Ungefähr benötigter Festplattenplatz: 43 MB

5.20.1. Installation von Gettext

Für die temporären Werkzeuge muss nur ein einziges Programm von Gettext erzeugt und installiert werden.

Bereiten Sie Gettext zum Kompilieren vor:

```
cd gettext-tools
./configure --prefix=/tools --disable-shared
```

Die Bedeutung des configure-Parameters:

--disable-shared
: Zu diesem Zeitpunkt müssen keine gemeinsamen Bibliotheken von Gettext installiert werden, daher müssen sie auch nicht kompiliert werden.

Kompilieren Sie das Paket:

```
make -C lib
make -C src msgfmt
```

Weil nur ein einziges Programm kompiliert wurde, kann die Testsuite nicht ausgeführt werden. Daher wird davon abgeraten, die Testsuite an diesem Punkt auszuführen.

Installieren Sie das Programm **msgfmt**:

```
cp -v src/msgfmt /tools/bin
```

Details zu diesem Paket finden Sie in Abschnitt 6.36.2, „Inhalt von Gettext“

5.21. Grep-2.5.1a

Das Paket Grep enthält Programme zum Durchsuchen von Dateien.

Geschätzte Kompilierzeit: 0.1 SBU
Ungefähr benötigter Festplattenplatz: 4.8 MB

5.21.1. Installation von Grep

Bereiten Sie Grep zum Kompilieren vor:

```
./configure --prefix=/tools \
    --disable-perl-regexp
```

Die Bedeutung des configure-Parameters:

--disable-perl-regexp
Dies stellt sicher, dass **grep** nicht gegen die PCRE-Bibliothek verlinkt wird. Diese Bibliothek könnte auf dem Host-System installiert sein, ist aber später in der **chroot**-Umgebung nicht mehr verfügbar.

Kompilieren Sie das Paket:

```
make
```

Um das Ergebnis zu prüfen, führen Sie **make check** aus.

Installieren Sie das Paket:

```
make install
```

Details zu diesem Paket finden Sie in Abschnitt 6.37.2, „Inhalt von Grep“

5.22. Gzip-1.3.5

Das Paket Gzip enthält Programme zum Komprimieren und Dekomprimieren von Dateien.

Geschätzte Kompilierzeit: weniger als 0.1 SBU
Ungefähr benötigter Festplattenplatz: 2.2 MB

5.22.1. Installation von Gzip

Bereiten Sie Gzip zum Kompilieren vor:

```
./configure --prefix=/tools
```

Kompilieren Sie das Paket:

```
make
```

Dieses Paket enthält keine Testsuite.

Installieren Sie das Paket:

```
make install
```

Details zu diesem Paket finden Sie in Abschnitt 6.39.2, „Inhalt von Gzip“

5.23. M4-1.4.4

M4 enthält einen Makroprozessor.

Geschätzte Kompilierzeit: weniger als 0.1 SBU
Ungefähr benötigter Festplattenplatz: 3 MB

5.23.1. Installation von M4

Bereiten Sie M4 zum Kompilieren vor:

```
./configure --prefix=/tools
```

Kompilieren Sie das Paket:

```
make
```

Um das Ergebnis zu prüfen, führen Sie **`make check`** aus.

Installieren Sie das Paket:

```
make install
```

Details zu diesem Paket finden Sie in Abschnitt 6.16.2, „Inhalt von M4“

5.24. Make-3.80

Das Paket Make enthält Werkzeuge zum Kompilieren von Software.

Geschätzte Kompilierzeit: 0.1 SBU
Ungefähr benötigter Festplattenplatz: 7.8 MB

5.24.1. Installation von Make

Bereiten Sie Make zum Kompilieren vor:

```
./configure --prefix=/tools
```

Kompilieren Sie das Paket:

```
make
```

Um das Ergebnis zu prüfen, führen Sie **make check** aus.

Installieren Sie das Paket:

```
make install
```

Details zu diesem Paket finden Sie in Abschnitt 6.44.2, „Inhalt von Make“

5.25. Patch-2.5.4

Das Paket Patch enthält ein Programm zum Erzeugen oder Modifizieren von Dateien indem eine sogenannte „Patch"-Datei angewendet wird. Einen „Patch" erzeugt man üblicherweise mit **diff** und er beschreibt in maschinenlesbarer Form die Unterschiede zwischen zwei Versionen einer Datei.

Geschätzte Kompilierzeit: weniger als 0.1 SBU
Ungefähr benötigter Festplattenplatz: 1.6 MB

5.25.1. Installation von Patch

Bereiten Sie Patch zum Kompilieren vor:

```
./configure --prefix=/tools
```

Kompilieren Sie das Paket:

```
make
```

Dieses Paket enthält keine Testsuite.

Installieren Sie das Paket:

```
make install
```

Details zu diesem Paket finden Sie in Abschnitt 6.48.2, „Inhalt von Patch"

5.26. Perl-5.8.8

Das Paket Perl enthält die Skriptsprache Perl (Practical Extraction and Report Language).

Geschätzte Kompilierzeit: 0.7 SBU
Ungefähr benötigter Festplattenplatz: 84 MB

5.26.1. Installation von Perl

Zuerst müssen Sie mit dem folgenden Patch ein paar festeingestellte Pfade zur C-Bibliothek anpassen:

```
patch -Np1 -i ../perl-5.8.8-libc-2.patch
```

Bereiten Sie Perl nun zum Kompilieren vor (passen Sie auf, dass Sie 'Data/Dumper Fcntl IO POSIX' richtig schreiben—das sind alles Buchstaben):

```
./configure.gnu --prefix=/tools -Dstatic_ext='Data/Dumper Fcntl IO POSIX'
```

Die Bedeutung der configure-Parameter:

-Dstatic_ext='Data/Dumper Fcntl IO POSIX'
Damit wird Perl angewiesen, die notwendigsten statischen Erweiterungen zu installieren, die im nächsten Kapitel für die Coreutils und die Glibc benötigt werden.

Aus diesem Paket müssen nur wenige Programme kompiliert werden:

```
make perl utilities
```

Auch wenn Perl eine Testsuite enthält, sollte sie zum jetzigen Zeitpunkt noch nicht ausgeführt werden. Es wurden nur Teile von Perl installiert und das Ausführen von **make test** würde bewirken, dass nun der Rest von Perl kompiliert werden würden. Das ist zu diesem Zeitpunkt völlig unnötig, die Testsuite kann im nächsten Kapitel ausgeführt werden.

Installieren Sie diese Werkzeuge und ihre Bibliotheken an die richtige Stelle:

```
cp -v perl pod/pod2man /tools/bin
mkdir -pv /tools/lib/perl5/5.8.8
cp -Rv lib/* /tools/lib/perl5/5.8.8
```

Details zu diesem Paket finden Sie in Abschnitt 6.22.2, „Inhalt von Perl“

5.27. Sed-4.1.5

Das Paket Sed enthält einen Stream-Editor.

Geschätzte Kompilierzeit: 0.1 SBU
Ungefähr benötigter Festplattenplatz: 6.1 MB

5.27.1. Installation von Sed

Bereiten Sie Sed zum Kompilieren vor:

```
./configure --prefix=/tools
```

Kompilieren Sie das Paket:

```
make
```

Um das Ergebnis zu prüfen, führen Sie **make check** aus.

Installieren Sie das Paket:

```
make install
```

Details zu diesem Paket finden Sie in Abschnitt 6.20.2, „Inhalt von Sed“

5.28. Tar-1.15.1

Das Paket Tar enthält ein Archivprogramm.

Geschätzte Kompilierzeit: 0.2 SBU
Ungefähr benötigter Festplattenplatz: 13.7 MB

5.28.1. Installation von Tar

Wenn Sie die Testsuite ausführen möchten, wenden Sie zuvor bitte den folgenden Patch an. Dadurch werden Fehler im Zusammenhang mit GCC-4.0.3 behoben:

```
patch -Np1 -i ../tar-1.15.1-gcc4_fix_tests-1.patch
```

Bereiten Sie Tar zum Kompilieren vor:

```
./configure --prefix=/tools
```

Kompilieren Sie das Paket:

```
make
```

Um das Ergebnis zu prüfen, führen Sie **make check** aus.

Installieren Sie das Paket:

```
make install
```

Details zu diesem Paket finden Sie in Abschnitt 6.53.2, „Inhalt von Tar“

5.29. Texinfo-4.8

Das Paket Texinfo enthält Programme zum Lesen, Schreiben und Konvertieren von Info-Seiten (Systemdokumentation).

Geschätzte Kompilierzeit: 0.2 SBU
Ungefähr benötigter Festplattenplatz: 16.3 MB

5.29.1. Installation von Texinfo

Bereiten Sie Texinfo zum Kompilieren vor:

```
./configure --prefix=/tools
```

Kompilieren Sie das Paket:

```
make
```

Um das Ergebnis zu prüfen, führen Sie **make check** aus.

Installieren Sie das Paket:

```
make install
```

Details zu diesem Paket finden Sie in Abschnitt 6.54.2, „Inhalt von Texinfo“

5.30. Util-linux-2.12r

Das Paket Util-linux enthält verschiedene Werkzeuge. Darunter befinden sich Programme zum Umgang mit Dateisystemen, Konsolen, Partitionen und (System-)Meldungen.

Geschätzte Kompilierzeit: less than 0.1 SBU
Ungefähr benötigter Festplattenplatz: 8.9 MB

5.30.1. Installation von Util-linux

Util-linux verwendet die gerade frisch installierten Header und Bibliotheken im Ordner `/tools` nicht automatisch. Korrigieren Sie dieses Problem indem Sie das configure-Skript anpassen:

```
sed -i 's@/usr/include@/tools/include@g' configure
```

Bereiten Sie Util-linux zum Kompilieren vor:

```
./configure
```

Kompilieren Sie einige unterstützende Routinen:

```
make -C lib
```

Aus diesem Paket müssen nur wenige Programme kompiliert werden:

```
make -C mount mount umount
make -C text-utils more
```

Dieses Paket enthält keine Testsuite.

Nun kopieren Sie diese Programme in unseren temporären Ordner tools:

```
cp mount/{,u}mount text-utils/more /tools/bin
```

Details zu diesem Paket finden Sie in Abschnitt 6.56.3, „Inhalt von Util-linux“

5.31. Stripping

Die Schritte in diesem Abschnitt sind optional. Wenn Ihre LFS-Partition sehr klein ist, werden Sie froh sein, ein paar unnötige Dinge loswerden zu können. Die bisher erstellten ausführbaren Dateien und Bibliotheken enthalten ungefähr 70 MB nicht benötigter Debugging-Symbole. So entfernen Sie diese Symbole:

```
strip --strip-debug /tools/lib/*
strip --strip-unneeded /tools/{,s}bin/*
```

Das erste der obigen Kommandos überspringt rund 20 Dateien mit der Meldung, dass der Dateityp nicht erkannt wurde. Die meisten dieser Dateien sind Skripte und keine Binärdateien.

Passen Sie auf, dass Sie *--strip-unneeded* *nicht* auf Bibliotheken anwenden — sie würden zerstört werden und dann müssten Sie die Toolchain neu kompilieren.

Um weitere 20 MB Platz zu sparen, können Sie die Dokumentation entfernen:

```
rm -rf /tools/{info,man}
```

Zum Kompilieren von Glibc benötigen Sie nun mindestens 850 MB freien Platz in `$LFS`. Wenn Sie Glibc kompilieren und installieren können, werden Sie mit den restlichen Paketen keine Platzprobleme bekommen.

5.32. Ändern des Besitzers



Anmerkung

Für den Rest des Buches sollten Sie als Benutzer `root` arbeiten, und nicht als `lfs`. An dieser Stelle sollten Sie außerdem nochmals überprüfen, ob `$LFS` korrekt eingestellt ist.

Im Augenblick gehört der Ordner `$LFS/tools` dem Benutzer `lfs`. Dieser existiert aber nur auf dem Host-System. Wenn Sie den Ordner `$LFS/tools` in seinem jetzigen Zustand behalten, gehören die Dateien einer Benutzer-ID zu der es kein Benutzerkonto gibt. Das ist gefährlich, denn ein später erstelltes Konto könnte genau diese ID erhalten und wäre damit der Besitzer von `$LFS/tools` und aller darin enthaltenen Dateien. Dieser Benutzer könnte alle Dateien unbemerkt manipulieren.

Um dieses Problem zu vermeiden, können Sie Ihrem LFS-System den Benutzer `lfs` später beim Erzeugen der `/etc/passwd` hinzufügen und ihm die gleiche Benutzer-ID und Gruppen-ID wie auf Ihrem Host-System geben. Besser ist es jedoch, jetzt den Benutzer `root` zum Besitzer des Ordners machen. Benutzen Sie dazu dieses Kommando:

```
chown -R root:root $LFS/tools
```

Auch wenn Sie `$LFS/tools` nach Fertigstellung dieses LFS löchen können, entscheiden Sie sich vielleicht, den Ordner dennoch aufzuheben. Dies kann z. B. sinnvoll sein, um weitere LFS-Systeme der *selben Buchversion* zu installieren. Wie Sie am besten eine Sicherungskopie von `$LFS/tools` erstellen, ist Ihnen als lehrreiches Experiment selber überlassen ;-)

Teil III. Installation des LFS-Systems

Kapitel 6. Installieren der grundlegenden System-Software

6.1. Einführung

In diesem Kapitel begeben Sie sich an den eigentlichen Ort des Geschehens und beginnen mit dem Bau des endgültigen LFS-Systems. Im einzelnen chroot'en Sie in Ihr temporäres Mini-Linux, erzeugen einige Hilfsmittel und beginnen dann, alle Pakete der Reihe nach zu installieren.

Die Installation der Software ist sehr gradlinig. Auch wenn die Installationsanweisungen an einigen Stellen sicherlich kürzer hätten ausfallen können, haben wir uns für die ausführliche Variante entschieden. Wenn Sie lernen möchten wie Linux intern funktioniert, dann sollten Sie wissen, wofür die jeweiligen Pakete benutzt werden und warum ein Benutzer oder das System auf sie angewiesen sind. Deshalb finden Sie zu jedem Paket eine Zusammenfassung seines Inhalts und eine kurze Beschreibung zu den installierten Programmen und Bibliotheken.

Falls Sie in diesem Kapitel Compiler-Optimierungen einsetzen möchten, lesen Sie bitte die Anleitung unter *http://www.linuxfromscratch.org/hints/downloads/files/optimization.txt*. Compiler-Optimierungen können ein Programm etwas schneller ablaufen lassen, aber sie können auch zu Schwierigkeiten beim Kompilieren oder Ausführen von Programmen führen. Wenn sich ein Paket nicht kompilieren lässt, versuchen Sie es erstmal ohne Optimierungen und schauen Sie, ob das Problem dann behoben ist. Selbst wenn das Paket mit Compiler-Optmierungen kompilierbar ist, besteht die Gefahr, dass es fehlerhaft kompiliert wurde (z. B. aufgrund der komplexen Zusammenhänge zwischen Code und den Compilerwerkzeugen). Beachten Sie auch, dass die Optionen `-march` und `-mtune` Schwierigkeiten mit den Paketen der Toolchain verursachen werden (Binutils, GCC und Glibc). Kurz gesagt, der potientelle Geschwindigkeitsvorteil wird durch das hohe Risiko aufgehoben. Wenn Sie das erste mal ein LFS installieren, sollten Sie keine Compiler-Optimierungen einsetzen. Ihr neues System wird dennoch sehr schnell und gleichzeitig auch noch stabil sein.

Die Installationsreihenfolge in diesem Kapitel muss auf jeden Fall eingehalten werden, sonst könnten einige Programme eventuell feste Referenzen auf `/tools` erhalten. *Kompilieren Sie aus diesem Grund auch nicht mehrere Pakete gleichzeitig*. Gleichzeitiges Kompilieren kann Ihnen eine Zeitersparnis bringen, besonders auf Mehrprozessormaschinen, aber es kann zu Programmen führen, die Referenzen auf `/tools` enthalten und nicht mehr funktionieren sobald dieser Ordner entfernt wird.

Auf jeder Informationsseite finden Sie zu Beginn ein paar allgemeine Informationen zum jeweiligen Paket: Eine kurze Beschreibung des Inhalts, eine Abschätzung der benötigten Kompilierzeit und des benötigten Festplattenspeichers beim Kompilieren. Nach den Installationsanweisungen folgt eine Liste der Programme und Bibliotheken (inklusive einer kurzen Beschreibung), die mit dem Paket installiert werden.

6.2. Vorbereiten der virtuellen Kernel-Dateisysteme

Verschiedene vom Kernel exportierte Dateisysteme werden für die Kommunikation zwischen dem Kernel selbst und dem sog. Userspace verwendet. Dies sind virtuelle Dateisysteme in Hinsicht darauf, dass sie keinen Speicherplatz auf der Festplatte verbrauchen. Der Inhalt der Dateisysteme liegt vollständig im Arbeitsspeicher.

Erstellen Sie die Ordner, in die dann die virtuellen Dateisysteme eingehängt werden:

```
mkdir -pv $LFS/{dev,proc,sys}
```

6.2.1. Erzeugen der wichtigsten Gerätedateien

Zum Booten des Kernel müssen nur wenige Gerätedateien vorhanden sein, im einzelnen `console` und `null`. Die Gerätedateien werden auf der Festplatte erzeugt, damit sie vor dem Start von **udevd** verfügbar sind (insbesondere wenn Linux mit dem Parameter *`init=/bin/bash`* gestartet wird). Erstellen Sie die Gerätedateien mit diesen Kommandos:

```
mknod -m 600 $LFS/dev/console c 5 1
mknod -m 666 $LFS/dev/null c 1 3
```

6.2.2. Einhängen und Füllen von /dev

Die empfohlene Vorgehensweise um `/dev` mit Gerätedateien zu füllen ist, in `/dev` ein virtuelles Dateisystem wie z. B. `tmpfs` einzuhängen und die Geräte dynamisch zu erzeugen, sobald sie erkannt oder verwendet werden. Die meisten Geräte werden beim Booten erkannt und von Udev erzeugt. Weil das neue System aber bislang noch nicht gebootet wurde, müssen Sie diese Arbeit erstmal selbst erledigen. Sie werden nun den `/dev`-Ordner des Host-Systems mit dern bind-Methode einhängen. Es handelt sich dabei um eine besondere Methode zum Einhängen eines Dateisystems, bei der ein Ordner oder Mountpunkt gespiegelt bzw. zusätzlich an einer weiteren Stelle des Dateisystems eingehängt wird. Benutzen Sie dazu das folgende Kommando:

```
mount --bind /dev $LFS/dev
```

6.2.3. Einhängen der virtuellen Kernel-Dateisysteme

Hängen Sie nun die verbleibenden virtuellen Kernel-Dateisysteme ein:

```
mount -vt devpts devpts $LFS/dev/pts
mount -vt tmpfs shm $LFS/dev/shm
mount -vt proc proc $LFS/proc
mount -vt sysfs sysfs $LFS/sys
```

6.3. Paketverwaltung

Paketverwaltung ist eine der am häufigsten nachgefragten Erweiterungen für das LFS-Buch. Mit einer Paketverwaltung können Sie die Installation von Dateien protokollieren und diese dann später leicht wieder deinstallieren oder Pakete aktualisieren. Vorab erstmal eine Klarstellung: NEIN—dieses Kapitel behandelt keine Paketverwaltung im Detail und wird Ihnen auch keine empfehlen. Sie werden hier nur Informationen zu den am weitesten verbreiteten Methoden und Techniken erhalten. Die für Sie perfekte Paketverwaltung könnte dabei sein, vielleicht ist es auch eine Kombination aus zwei oder mehr Techniken.

Einige Gründe, warum weder in LFS noch in BLFS eine Paketverwaltung installiert wird, sind:

- Der Umgang mit einer Paketverwaltung lenkt die Aufmerksamkeit vom eigentlichen Ziel des Buches ab—nämlich zu lernen, wie man ein Linux-System von Hand erstellt.
- Es gibt viele Paketverwaltungen; jede hat ihre Vor- und Nachteile. Es ist schwierig, eine zu finden, die alle Leser zufriedenstellen würde.

Es wurden einige Tipps zu diesem Thema geschrieben. Lesen Sie im *Hints-Projekt* nach, vielleicht finden Sie eine passende Paketverwaltung für Sie.

6.3.1. Aktualisierung von Paketen

Mit einer Paketverwaltung ist es recht einfach, ein Paket zu aktualisieren. Grundsätzlich kann man aber auch die Anleitungen in LFS und BLFS zur Aktualisierung auf neuere Versionen verwenden. Im Folgenden finden Sie allerdings ein paar wichtige Dinge, die Sie beim Aktualisierungen von Programmen beachten sollten (insbesondere auf einem laufenden System).

- Wenn eines der Toolchain-Pakete (Glibc, GCC oder Binutils) auf eine neue Minor-Version aktualisiert werden muss, ist es meist besser, LFS neu zu installieren. Es ist *möglich*, dass einfaches Neuinstallieren der betroffenen Pakete in der richtigen Abhängigkeitsreihenfolge ausreicht, aber davon wird dringend abgeraten! Wenn Sie also z. B. glibc-2.2.x auf glibc-2.3.x aktualisieren müssen, sollten Sie neu installieren. Die Aktualisierung innerhalb einer Mikro-Version ist normalerweise problemlos möglich, wenn auch nicht zu 100% garantiert. Beispielsweise sollte ein Versionsupdate von glibc-2.3.4 auf glibc-2.3.5 keine Schwierigkeiten bereiten.
- Wenn Sie ein Paket aktualisieren, das gemeinsam verwendete Bibliotheken enthält und sich mit der Aktualisierung der Name der Bibliothek ändert, dann müssen alle Programme, die die Bibliothek verwenden, neu kompiliert werden. (Beachten Sie: zwischen dem Namen der Bibliothek und der Paketversion besteht grundsätzlich kein Zusammenhang.) Angenommen Sie haben das Paket foo-1.2.3 mit der gemeinsamen Bibliothek `libfoo.so.1`. Dieses Paket aktualisieren Sie nun auf Version 1.2.4, welche die Bibliothek namens `libfoo.so.2` installiert. In diesem Fall müssen Sie alle Programme neu kompilieren, die `libfoo.so.1` verwenden, damit sie in Zukunft `libfoo.so.2` referenzieren. Beachten Sie auch: Sie dürfen die alte Bibliothek erst entfernen, wenn alle davon abhängigen Pakete aktualisiert wurden.

6.3.2. Techniken zur Paketverwaltung

Im Folgenden werden einige Techniken zur Paketverwaltung beschrieben. Bevor Sie sich für eine entscheiden, informieren Sie sich bitte über die jeweilige Technik, insbesondere über die möglichen Nachteile.

6.3.2.1. Ich behalte alles im Kopf!

Ja, auch das ist eine Methode der Paketverwaltung. Manche Leute benötigen einfach keine Software zur Paketverwaltung, weil sie alle Pakete gut kennen und wissen, welche Dateien vom jeweiligen Paket installiert werden. Andere Leute benötigen möglicherweise keine Paketverwaltung, weil sie LFS neu installieren, sobald ein Paket geändert wird.

6.3.2.2. Installation in separate Ordner

Diese einfache Methode der Paketverwaltung benötigt keine weitere Software. Jedes Paket wird einfach in einen eigenen Ordner installiert. Beispielsweise wird foo-1.1 in den Ordner `/usr/pkg/foo-1.1` installiert und dann einen symbolischen Link von `/usr/pkg/foo` nach `/usr/pkg/foo-1.1` angelegt. Wenn später auf die neuere Version foo-1.2 aktualisiert wird, so erfolgt die Installation in den Ordner `/usr/pkg/foo-1.2` und der symbolischen Link wird einfach durch einen neuen ersetzt.

Umgebungsvariablen wie `PATH`, `LD_LIBRARY_PATH`, `MANPATH`, `INFOPATH` und `CPPFLAGS` müssen so angepasst werden, dass Sie `/usr/pkg/foo` enthalten. Diese Methode wird sehr unhandlich, wenn auf diese Weise viele Softwarepakete verwaltet werden sollen.

6.3.2.3. Paketverwaltung mit symbolischen Links

Es handelt sich hierbei im Grunde nur um eine Variation der vorigen Paketverwaltungs-Technik. Jedes Paket wird genauso installiert wie zuvor beschrieben. Anstatt jedoch den ganzen Ordner mit einem Symlink zu versehen, wird für jede einzelne Datei eine Verknüpfung in `/usr` angelegt. Auf diese Weise müssen die Umgebungsvariablen nicht angepasst werden. Die vielen symbolischen Verknüpfungen können natürlich vom Benutzer selbst angelegt werden, jedoch gibt es auch einige Programme dafür, die diese Technik verwenden: Stow, Epkg, Graft und Depot sind einige Beispiele.

Die Installation muss allerdings so angepasst werden, so dass das Paket "denkt", es wäre in `/usr` installiert, obwohl die Dateien tatsächlich in `/usr/pkg` gespeichert werden. Das vortäuschen einer solchen Installation ist manchmal nicht ganz leicht. Nehmen wir an, Sie möchten das Paket libfoo-1.1 installieren. Die folgenden Kommandos würden das Paket nicht korrekt installieren:

```
./configure --prefix=/usr/pkg/libfoo/1.1
make
make install
```

Die Installation ansich wird funktionieren, aber die abhängigen Pakete werden nicht korrekt auf libfoo verweisen. Wenn Sie ein Paket kompilieren, welches libfoo benötigt, so wird es gegen `/usr/pkg/libfoo/1.1/lib/libfoo.so.1` linken, anstatt den korrekten Pfad `/usr/lib/libfoo.so.1` zu verwenden. Der korrekte Ansatz ist der Einsatz der Variable `DESTDIR`, mit der die Installation in einen anderen Ordner vorgetäuscht werden kann. Dies funktioniert wie folgt:

```
./configure --prefix=/usr
make
make DESTDIR=/usr/pkg/libfoo/1.1 install
```

Diese Methode funktioniert mit den meisten Softwarepaketen, aber leider nicht mit allen. Die inkompatiblen Pakete müssen Sie entweder von Hand installieren, oder Sie installieren sie unterhalb von `/opt`.

6.3.2.4. Paketverwaltung mittels Zeitstempel

Bei dieser Technik wird jede Datei vor der Installation mit einem Zeitstempel versehen. Nach der Installation können alle installierten Dateien mit einem einfachen **find**-Kommando gefunden und protokolliert werden. Die Paketverwaltung "install-log" setzt diese Methode ein.

Obwohl diese Methode natürlich sehr einfach ist, hat sie leider zwei Nachteile. Wenn während der Installation Dateien ohne oder mit einem anderen Zeitstempel als der aktuellen Zeit installiert werden, so wird deren Installation nicht protokolliert. Desweiteren kann diese Methode nur funktionieren, wenn maximal ein Paket zur gleichen Zeit installiert wird. Das Protokoll ist nicht mehr zuverlässig, wenn z. B. auf einer anderen Konsole ein weiteres Programm zeitgleich installiert wird.

6.3.2.5. Paketverwaltung mittels LD_PRELOAD

Bei diesem Ansatz wird vor der Installation eine Bibliothek vorgeladen. Während der Installation protokolliert diese Bibliothek alle Installationsvorgänge mit, indem sie sich an verschiedene ausführbare Programme wie **cp**, **install** und **mv** ahängt und die Systemaufrufe mitverfolgt. Damit dies funktionieren kann, müssen alle ausführbaren Programme dynamisch verlinkt und weder mit dem suid- noch dem sgid-Bit versehen sein. Das Vorladen der Bibliothek kann unter Umständen auch Nebeneffekte bei der Installation hervorrufen. Deshalb sollten Sie diese Methode ausführlich testen, bevor Sie sie produktiv einsetzen.

6.3.2.6. Paket-Archive erstellen

Bei dieser Methode wird die Installation in einem separaten Unterordner vorgenommen, ähnlich wie bei der Methode mit symbolischen Verknüpfungen. Nach der Installation wird aus der Ordnerstruktur ein Archiv mit den installierten Dateien erzeugt. Dieses Archiv kann dann zur Installation benutzt werden. Auf diese Weise können Sie ein Archiv auch auf mehreren Rechnern installieren.

Diese Methode kommt in den meisten kommerziellen Distributionen zum Einsatz. Beispiele für Paketverwaltungen, die diese Methode einsetzen, sind: RPM (welches im Übrigen von der *Linux Standard Base Spezifikation* erfordert wird), pkg-utils, Debians apt und Gentoos Portage-System. Eine Anleitung zur Verwendung dieses Paketverwaltungs-Systems finden Sie unter *http://www.linuxfromscratch.org/hints/downloads/files/fakeroot.txt*.

6.3.2.7. Benutzerbasierte Paketverwaltung

Diese für LFS einmalige Methode hat sich Matthias Benkmann ausgedacht. Informationen dazu finden Sie im *Hints Projekt*. Bei der Benuterbasierten Paketverwaltung wird jedes Paket unter Verwendung einer eigenen Benutzer-ID an den Standard-Installationsort installiert. Alle zu einem Paket gehörenden Dateien können anhand der Benutzer-ID leicht wiedergefunden werden. Die Vor- und Nachteile dieser Paketverwaltung sind allerdings so umfangreich, dass wir sie hier in diesem Kapitel nicht alle beschreiben können. Alle notwendigen Informationen finden Sie unter *http://www.linuxfromscratch.org/hints/downloads/files/more_control_and_pkg_man.txt*.

6.4. Betreten der chroot-Umgebung

Es ist nun an der Zeit, die chroot-Umgebung zu betreten und mit der Installation der benötigten Pakete zu beginnen. Immer noch als `root` führen Sie das folgende Kommando aus. Damit betreten Sie die neue kleine Welt, die zur Zeit nur mit temporären Werkzeugen ausgestattet ist:

```
chroot "$LFS" /tools/bin/env -i \
    HOME=/root TERM="$TERM" PS1='\u:\w\$ ' \
    PATH=/bin:/usr/bin:/sbin:/usr/sbin:/tools/bin \
    /tools/bin/bash --login +h
```

Die an **env** übergebene Option *-i* löscht alle Variablen in der chroot-Umgebung. Danach werden nur die Variablen `HOME`, `TERM`, `PS1` und `PATH` wieder gesetzt. *TERM=$TERM* setzt die Variable `TERM` in der chroot-Umgebung auf den gleichen Wert wie außerhalb von chroot, diese Variable wird für das korrekte Funktionieren von Programmen wie **vim** und **less** benötigt. Wenn Sie weitere Variablen wie `CFLAGS` oder `CXXFLAGS` benötigen, ist dies ein guter Platz, um sie erneut zu setzen.

Von nun an brauchen Sie die Variable `LFS` nicht mehr, denn alle weiteren Befehle sind auf Ihr LFS beschränkt. Das was die laufende Shell für den Ordner `/` hält, ist in Wirklichkeit der Wert von `$LFS`, den Sie **chroot** oben als Parameter übergeben haben.

Beachten Sie, dass `/tools/bin` am Ende der Variable `PATH` steht. Das bewirkt, dass ein temporäres Werkzeug nicht mehr benutzt wird, sobald seine endgültige Version installiert ist. Zumindest, wenn die Shell sich nicht die Standorte von ausführbaren Dateien merkt—aus diesem Grund wird die Hash-Funktion der **bash** mit der Option *+h* abgeschaltet.

Die Eingabeaufforderung der **Bash** wird `I have no name!` ausgeben. Das ist normal und liegt daran, dass die Datei `/etc/passwd` derzeit noch fehlt. Mit Hilfe dieser Datei findet nämlich auch die Zuordnung von Benutzer-IDs zu Benutzernamen statt.



Anmerkung

Sie müssen alle Kommandos in den folgenden Kapiteln in der chroot-Umgebung ausführen. Wenn Sie die chroot-Umgebung aus irgendeinem Grund verlassen müssen (zum Beispiel wegen einem Neustart), dann denken Sie daran, die virtuellen Kernel-Dateisysteme wie in Kapitel Abschnitt 6.2.2, „Einhängen und Füllen von /dev“ und Abschnitt 6.2.3, „Einhängen der virtuellen Kernel-Dateisysteme“ erneut einzubinden *und* die chroot-Umgebung wieder zu betreten, bevor Sie mit der Installation fortfahren.

6.5. Erstellen der Ordnerstruktur

Nun bringen Sie ein wenig Struktur in das LFS-Dateisystem. Erzeugen Sie mit dem folgenden Kommando eine standardkonforme Ordnerstruktur:

```
mkdir -pv /{bin,boot,etc/opt,home,lib,mnt,opt}
mkdir -pv /{media/{floppy,cdrom},sbin,srv,var}
install -dv -m 0750 /root
install -dv -m 1777 /tmp /var/tmp
mkdir -pv /usr/{,local/}{bin,include,lib,sbin,src}
mkdir -pv /usr/{,local/}share/{doc,info,locale,man}
mkdir -v  /usr/{,local/}share/{misc,terminfo,zoneinfo}
mkdir -pv /usr/{,local/}share/man/man{1..8}
for dir in /usr /usr/local; do
  ln -sv share/{man,doc,info} $dir
done
mkdir -v /var/{lock,log,mail,run,spool}
mkdir -pv /var/{opt,cache,lib/{misc,locate},local}
```

Normalerweise werden Ordner in der Voreinstellung mit den Rechten 755 erzeugt, aber das ist nicht bei allen Ordnern erwünscht. Nehmen Sie bitte zwei Änderungen vor: eine für den Persönlichen Ordner von root und eine weitere an den Ordnern für temporäre Dateien.

Die erste Rechteänderung bewirkt, dass nicht jeder den Ordner /root betreten darf—das gleiche würde ein normaler Benutzer mit seinem Persönlichen Ordner auch tun. Die zweite Änderung sorgt dafür, dass jeder Benutzer in die Ordner /tmp und /var/tmp schreiben, aber nicht die Dateien anderer Benutzer löschen kann. Letzteres wird durch das „sticky bit“ bewirkt—dem höchsten Bit (1) in der Bit-Maske 1777.

6.5.1. Anmerkung zur FHS-Konformität

Unsere Ordnerstruktur basiert auf dem FHS-Standard (siehe *http://www.pathname.com/fhs/*). Desweiteren erzeugen wir aus Kompatibilitätsgründen symbolische Verknüpfungen für die Ordner man, doc und info. Viele Programm versuchen leider immer noch, ihre Dokumentation nach /usr/<ordner> oder /usr/local/<ordner> anstelle von /usr/share/<ordner> bzw. /usr/local/share/<ordner> zu installieren.Zusätzlich zu den oben erstellten Ordnern sieht der FHS-Standard auch das Vorhandensein von /usr/local/games und /usr/share/games vor. Zur Struktur in /usr/local/share macht FHS keine präzisen Angaben, daher haben wir nur die Ordner erstellt, die wir für nötig halten.

6.6. Erstellen notwendiger Dateien und symbolischer Verknüpfungen

Einige Programme haben fest eingestellte Pfade zu Programmen, die zum jetzigen Zeitpunkt aber noch nicht existieren. Deshalb erstellen Sie eine Reihe symbolischer Links, die im weiteren Verlauf des Kapitels beim Installieren der restlichen Software durch echte Dateien ersetzt werden.

```
ln -sv /tools/bin/{bash,cat,grep,pwd,stty} /bin
ln -sv /tools/bin/perl /usr/bin
ln -sv /tools/lib/libgcc_s.so{,.1} /usr/lib
ln -sv bash /bin/sh
```

Ein korrekt eingerichtetes Linux hält in `/etc/mtab` eine Liste der derzeit eingebundenen Dateisysteme vor. Ist die Datei nicht vorhanden, so wird sie beim ersten Einbinden eines Dateisystems automatisch erzeugt. Da wir aber innerhalb der chroot-Umgebung keine Dateisysteme einbinden werden, müssen wir die Datei selbst erstellen, weil einige Programme deren Vorhandensein voraussetzen:

```
touch /etc/mtab
```

Damit `root` sich am System anmelden kann und damit der Name „root“ der richtigen Benutzer-ID zugeordnet werden kann, müssen die entsprechenden Einträge in `/etc/passwd` und `/etc/group` vorhanden sein.

Erzeugen Sie `/etc/passwd` mit dem folgenden Kommando:

```
cat > /etc/passwd << "EOF"
root:x:0:0:root:/root:/bin/bash
EOF
```

Das tatsächliche Passwort für `root` (Das „x“ ist hier nur Platzhalter) wird erst später gesetzt.

Erstellen Sie `/etc/group` mit dem folgenden Kommando:

```
cat > /etc/group << "EOF"
root:x:0:
bin:x:1:
sys:x:2:
kmem:x:3:
tty:x:4:
tape:x:5:
daemon:x:6:
floppy:x:7:
disk:x:8:
lp:x:9:
dialout:x:10:
audio:x:11:
video:x:12:
utmp:x:13:
usb:x:14:
cdrom:x:15:
EOF
```

Die erzeugten Gruppen sind nicht Teil irgendeines Standards—es sind die Gruppen, die Udev in diesem Kapitel benutzt. Neben der Gruppe `root` mit der GID 0 schlägt die LSB (*Linux Standard Base*) nur die Gruppe `bin` mit der GID 1 vor. Alle anderen Gruppennamen und GIDs können durch den Anwender frei gewählt werden, weil gut geschriebene Pakete sich nicht auf GID-Nummern verlassen sollten, sondern den Gruppennamen verwenden.

Um die Meldung „I have no name!“ loszuwerden, starten Sie eine neue Shell. Die Auflösung von Benutzer- und Gruppennamen funktioniert sofort nach dem Erstellen von `/etc/passwd` und `/etc/group`, weil Sie in Kapitel 5 eine vollständige Glibc installiert haben.

```
exec /tools/bin/bash --login +h
```

Beachten Sie die Option *+h*. Durch sie wird das interne Pfad-Hashing der **Bash** abgeschaltet. Ohne diese Anweisung würde sich **bash** die Pfade zu ausführbaren Dateien merken und wiederverwenden. Weil die frisch installierten Programme aber sofort nach deren Installation an ihrem neuen Ort genutzt werden sollen, schalten Sie die Funktion für dieses Kapitel aus.

Die Programme **login**, **agetty**, und **init** (und einige weitere) verwenden Logdateien zum Protokollieren von Informationen. Dazu gehört z. B. wer sich zu welcher Zeit an das System angemeldet hat. Diese Programme protokollieren aber nur, wenn die entsprechenden Logdateien bereits existieren. Daher müssen Sie die Logdateien nun anlegen und die richtigen Recht vergeben:

```
touch /var/run/utmp /var/log/{btmp,lastlog,wtmp}
chgrp -v utmp /var/run/utmp /var/log/lastlog
chmod -v 664 /var/run/utmp /var/log/lastlog
```

Die Logdateien haben folgenden Zweck: `/var/run/utmp` protokolliert zur Zeit angemeldete Benutzer. `/var/log/wtmp` protokolliert alle An- und Abmeldungen. `/var/log/lastlog` protokolliert die letzte Anmeldung für jeden Benutzer. `/var/log/btmp` protokolliert fehlgeschlagene Anmeldeversuche.

6.7. Linux-Libc-Header-2.6.12.0

Das Paket Linux-Libc-Header enthält die „bereinigten“ Header-Dateien des Linux-Kernels

Geschätzte Kompilierzeit: weniger als 0.1 SBU
Ungefähr benötigter Festplattenplatz: 27 MB

6.7.1. Installation von Linux-Libc-Header

Über Jahre hinweg war es gängige Praxis, die „rohen“ Kernel-Header (direkt aus dem Kernel-Archiv) in `/usr/include` zu benutzen. Aber in den letzten Jahren haben die Kernel-Entwickler die Haltung eingenommen, dass man dies nicht tun sollte. Daraus entstand das Projekt Linux-Libc-Header. Es wurde entworfen um eine konsistente Version der Kernel-Header Programmierschnittstelle (API) zu bewahren.

Fügen Sie einen Userspace-Header und syscall hinzu, um die in neueren Linux-Versionen verfügbare Funktion "inotify" zu unterstützen:

```
patch -Np1 -i ../linux-libc-headers-2.6.12.0-inotify-3.patch
```

Installieren Sie die Header-Dateien:

```
install -dv /usr/include/asm
cp -Rv include/asm-i386/* /usr/include/asm
cp -Rv include/linux /usr/include
```

Stellen Sie sicher, dass die Header im Besitz von root sind:

```
chown -Rv root:root /usr/include/{asm,linux}
```

Stellen Sie sicher, dass normale Benutzer Leserechte auf die Header haben:

```
find /usr/include/{asm,linux} -type d -exec chmod -v 755 {} \;
find /usr/include/{asm,linux} -type f -exec chmod -v 644 {} \;
```

6.7.2. Inhalt von Linux-Libc-Header

Installierte Header: /usr/include/{asm,linux}/*.h

Kurze Beschreibungen

`/usr/include/{asm,linux}/*.h` Diese Dateien bilden die Linux Header-API.

6.8. Man-pages-2.34

Das Paket Man-pages enthält über 1.200 Hilfeseiten.

Geschätzte Kompilierzeit: weniger als 0.1 SBU
Ungefähr benötigter Festplattenplatz: 18.4 MB

6.8.1. Installation der Man-pages

Installieren Sie die Man-pages durch Ausführen von:

```
make install
```

6.8.2. Inhalt von Man-pages

Installierte Dateien: verschiedene Hilfeseiten (Man-pages)

Kurze Beschreibungen

`Hilfeseiten` Sie beschreiben z. B. Funktionen der Programmiersprache C und wichtige Geräte- und Konfigurationsdateien.

6.9. Glibc-2.3.6

Glibc enthält die C-Bibliothek. Sie stellt Systemaufrufe und grundlegende Funktionen zur Verfügung (z. B. das Zuweisen von Speicher, Durchsuchen von Ordnern, Öffnen und Schließen sowie Schreiben von Dateien, Zeichenkettenverarbeitung, Mustererkennung, Arithmetik etc.)

Geschätzte Kompilierzeit: 13.5 SBU inkl. Testsuite
Ungefähr benötigter Festplattenplatz: 510 MB inkl. Testsuite

6.9.1. Installation von Glibc



Anmerkung

Einige Pakete außerhalb von LFS empfehlen, die GNU-Software libiconv zu installieren, um Daten von einer Kodierung in eine andere umzuwandeln. Auf der Webseite des Projektes unter *http://www.gnu.org/software/libiconv/* wird gesagt: „This library provides an `iconv()` implementation, for use on systems which don't have one, or whose implementation cannot convert from/to Unicode.“ Glibc enthält eine `iconv()`-Funktion und kann auch von/nach Unicode konvertieren, deshalb wird libiconv auf einem LFS-System nicht benötigt.

Das Installationssystem der Glibc ist sehr eigenständig und lässt sich perfekt installieren, selbst wenn die specs-Datei unseres Compilers und der Linker immer noch auf `/tools` verweisen. Sie können die specs-Datei und den Linker nicht vor der Installation von Glibc modifizieren, weil die Autoconf-Tests von Glibc dann falsche Resultate ergeben würden.

Das Tar-Archiv glibc-libidn fügt der Glibc Unterstützung für länderspezifische Domänennamen hinzu (IDN). Viele Programme, die IDN unterstützen, benötigen allerdings die vollständige Bibliothek `libidn` (siehe *http://www.linuxfromscratch.org/blfs/view/svn/general/libidn.html*), und nicht diese Erweiterung. Entpacken Sie das Tar-Archiv innerhalb des Glibc-Quellordners:

```
tar -xf ../glibc-libidn-2.3.6.tar.bz2
```

Dieser Patch behebt einige Compilerfehler für Pakete, die `linux/types.h` nach `sys/kd.h` einbinden:

```
patch -Np1 -i ../glibc-2.3.6-linux_types-1.patch
```

Fügen Sie einen Header hinzu, um syscall-Funktionen für die Funktion "inotify" in neueren Linux-Kerneln hinzuzufügen:

```
patch -Np1 -i ../glibc-2.3.6-inotify-1.patch
```

Unter Verwendung der locale vi_VN.TCVN verleibt die **bash** beim Start in einer Endlosschleife. Ob dies ein Fehler der **bash** oder von Glibc ist, ist derzeit nicht bekannt. Verhindern Sie das Problem, indem Sie diese locale von der Installation ausschließen:

```
sed -i '/vi_VN.TCVN/d' localedata/SUPPORTED
```

Während **make install** läuft, wird ein Skript namens `test-installation.pl` ausgeführt, welches die neu installierte Glibc überprüft. Unsere Toolchain zeigt jedoch noch auf den Ordner `/tools`, woraufhin die falsche Glibc getestet werden würde. Sie können das Skript zwingen, die richtige Glibc zu testen. Verwenden Sie dazu dieses Kommando:

```
sed -i \
's|libs -o|libs -L/usr/lib -Wl,-dynamic-linker=/lib/ld-linux.so.2 -o|' \
        scripts/test-installation.pl
```

Die Dokumentation von Glibc empfiehlt, zum Kompilieren einen gesonderten Ordner zu verwenden:

```
mkdir -v ../glibc-build
cd ../glibc-build
```

Bereiten Sie Glibc zum Kompilieren vor:

```
../glibc-2.3.6/configure --prefix=/usr \
    --disable-profile --enable-add-ons \
    --enable-kernel=2.6.0 --libexecdir=/usr/lib/glibc
```

Die Bedeutung der neuen Parameter zu configure:

--libexecdir=/usr/lib/glibc
Dadurch wird das Programm **pt_chown** in `/usr/lib/glibc` anstelle von `/usr/libexec` installiert.

Kompilieren Sie das Paket:

```
make
```



Wichtig

In diesem Abschnitt wird die Testsuite von Glibc als *absolut kritisch* betrachtet. Sie sollten diesen Schritt *unter keinen Umständen überspringen*.

Testen Sie das Ergebnis:

```
make -k check 2>&1 | tee glibc-check-log
grep Error glibc-check-log
```

Wahrscheinlich werden Sie einen erwarteten (ignorierten) Fehler im *posix/annexc*-Test bemerken. Desweiteren ist die Glibc-Testsuite ein wenig vom Host-System abhängig. Dies ist eine Liste der häufigsten Fehler:

- Die Tests *nptl/tst-clock2* und *tst-attr3* schlagen manchmal fehl. Der Grund dafür ist nicht völlig klar; die Ursache könnte mit hoher Systemlast zusammenhängen.
- Der math-Test schlägt manchmal fehl, wenn Sie ein System mit einer älteren Intel- oder AMD-CPU verwenden.
- Falls Sie die LFS-Partition mit der Option *`noatime`* eingehängt haben, wird der *atime*-Test fehlschlagen. Wie schon unter Abschnitt 2.4, „Einhängen (mounten) der neuen Partition“ erwähnt wurde, sollten Sie den Parameter *`noatime`* beim Bau von LFS nicht verwenden.
- Auf alter oder langsamer Hardware oder unter hoher Systemlast können einige Tests aufgrund von Zeitüberschreitungen fehlschlagen.

Auch wenn es nur eine harmlose Meldung ist, die Installationsroutine von Glibc wird sich über die fehlende Datei `/etc/ld.so.conf` beschweren. Beheben Sie diese störende Warnung mit:

```
touch /etc/ld.so.conf
```

Installieren Sie das Paket:

```
make install
```

Installieren Sie den inotify-Header zu den anderen System-Header-Dateien:

```
cp -v ../glibc-2.3.6/sysdeps/unix/sysv/linux/inotify.h \
    /usr/include/sys
```

Die Locales, mit deren hilfe Systemmeldungen in Ihrer Sprache ausgegeben werden können, wurden durch das obige Kommando nicht mitinstalliert. Diese Locales werden nicht unbedingt benötigt, jedoch würden einige Testsuites der noch folgenden Pakete ohne sie ein paar wichtige Tests überspringen.

Mit dem Kommando **localedef** können Sie auch einen individuellen Satz an Locales installieren. Das erste untenstehenden Kommando kombiniert beispielsweise die Zeichensatz-unabhängige Localedefinition `/usr/share/i18n/locales/de_DE` mit der Zeichensatzdefinition `/usr/share/i18n/charmaps/ISO-8859-1.gz` und fügt das Ergebnis an `/usr/lib/locale/locale-archive` an. Mit den folgenden Kommandos erstellen Sie einen Minimalen Satz Locales, die für die kommenden Testsuites benötigt werden:

```
mkdir -pv /usr/lib/locale
localedef -i de_DE -f ISO-8859-1 de_DE
localedef -i de_DE@euro -f ISO-8859-15 de_DE@euro
localedef -i en_HK -f ISO-8859-1 en_HK
localedef -i en_PH -f ISO-8859-1 en_PH
localedef -i en_US -f ISO-8859-1 en_US
localedef -i en_US -f UTF-8 en_US.UTF-8
localedef -i es_MX -f ISO-8859-1 es_MX
localedef -i fa_IR -f UTF-8 fa_IR
localedef -i fr_FR -f ISO-8859-1 fr_FR
localedef -i fr_FR@euro -f ISO-8859-15 fr_FR@euro
localedef -i fr_FR -f UTF-8 fr_FR.UTF-8
localedef -i it_IT -f ISO-8859-1 it_IT
localedef -i ja_JP -f EUC-JP ja_JP
```

Installieren Sie zudem auch noch die Locale für Ihr Land, Ihre Sprache und Ihren Zeichensatz.

Alternativ können Sie auch alle Locales auf einmal installieren, die in `glibc-2.3.6/localedata/SUPPORTED` aufgelistet werden (die Liste enthält die obigen Locales und noch viele weitere). Dieses Kommando benötigt allerdings ein wenig Zeit:

```
make localedata/install-locales
```

Im unwahrscheinlichen Fall, dass Sie noch weitere Locales benötigen, verwenden Sie das Kommando **localedef**, um die nicht in `glibc-2.3.6/localedata/SUPPORTED` gelisteten Locales zu installieren.

6.9.2. Einrichten von Glibc

Sie müssen die Datei `/etc/nsswitch.conf` erstellen. Glibc gibt zwar Standardwerte vor wenn die Datei fehlt oder beschädigt ist, aber diese funktionieren nicht besonders gut mit Netzwerken. Außerdem müssen Sie die Zeitzone korrekt einstellen.

Erstellen Sie nun die Datei `/etc/nsswitch.conf`:

```
cat > /etc/nsswitch.conf << "EOF"
# Begin /etc/nsswitch.conf

passwd: files
group: files
shadow: files

hosts: files dns
networks: files

protocols: files
services: files
ethers: files
rpc: files

# End /etc/nsswitch.conf
EOF
```

Mit diesem Skript finden Sie heraus, in welcher Zeitzone Sie sich befinden:

```
tzselect
```

Nachdem Sie ein paar Fragen zu Ihrem Standort beantwortet haben, wird das Skript den Namen Ihrer Zeitzone ausgeben. Die Ausgabe könnte z. B. *Europe/Berlin* lauten.

Erstellen Sie dann mit dem folgenden Kommando die Datei `/etc/localtime`:

```
cp -v --remove-destination /usr/share/zoneinfo/<xxx> \
    /etc/localtime
```

Anstelle von `<xxx>` müssen Sie natürlich den Namen der Zeitzone einsetzen, der Ihnen von **tzselect** ausgegeben wurde (z. B. Europe/Berlin).

Die Bedeutung der Option zu cp:

`--remove-destination`
: Dadurch wird das Entfernen des bereits vorhandenen symbolischen Links erzwungen. Sie ersetzen den Link durch eine Kopie der echten Datei, weil wir den Fall abdecken wollen, das `/usr` auf einer separaten Partition liegen könnte. Das würde z. B. dann problematisch werden, wenn der Single-User-Modus gebootet wird.

6.9.3. Einrichten des dynamischen Laders

Per Voreinstellung sucht der dynamische Lader (`/lib/ld-linux.so.2`) in `/lib` und `/usr/lib` nach den dynamischen Bibliotheken, die zur Laufzeit von ausführbaren Programmen benötigt werden. Wenn die benötigten Bibliotheken allerdings außerhalb von `/lib` und `/usr/lib` liegen, müssen Sie diese Ordner in `/etc/ld.so.conf` eintragen, damit der dynamische Lader sie finden kann. Zwei Ordner sind dafür bekannt, weitere Bibliotheken zu enthalten: `/usr/local/lib` und `/opt/lib`. Diese Ordner fügen Sie gleich mit in den Suchpfad ein.

Erstellen Sie die neue Datei `/etc/ld.so.conf` mit dem folgenden Kommando:

```
cat > /etc/ld.so.conf << "EOF"
# Begin /etc/ld.so.conf

/usr/local/lib
/opt/lib

# End /etc/ld.so.conf
EOF
```

6.9.4. Inhalt von Glibc

Installierte Programme: catchsegv, gencat, getconf, getent, iconv, iconvconfig, ldconfig, ldd, lddlibc4, locale, localedef, mtrace, nscd, nscd_nischeck, pcprofiledump, pt_chown, rpcgen, rpcinfo, sln, sprof, tzselect, xtrace, zdump und zic
Installierte Bibliotheken: ld.so, libBrokenLocale.{a,so}, libSegFault.so, libanl.{a,so}, libbsd-compat.a, libc.{a,so}, libcidn.so, libcrypt.{a,so}, libdl.{a,so}, libg.a, libieee.a, libm.{a,so}, libmcheck.a, libmemusage.so, libnsl.a, libnss_compat.so, libnss_dns.so, libnss_files.so, libnss_hesiod.so, libnss_nis.so, libnss_nisplus.so, libpcprofile.so, libpthread.{a,so}, libresolv.{a,so}, librpcsvc.a, librt.{a,so}, libthread_db.so, and libutil.{a,so}

Kurze Beschreibungen

catchsegv	Kann zum Erzeugen eines Stacktrace benutzt werden (falls ein Programm mit einem Speicherzugriffsfehler abstürzt).
gencat	Erzeugt Nachrichtenkataloge.
getconf	Zeigt System-Konfigurationswerte für dateisystemspezifische Variablen an.
getent	Liest Einträge aus einer administrativen Datenbank.
iconv	Führt Zeichensatzkonvertierungen durch.
iconvconfig	Erzeugt schnellladende **iconv** Konfigurationsdateien.
ldconfig	Richtet die Laufzeitbindungen des dynamischen Linkers ein.
ldd	Gibt aus, welche gemeinsamen Bibliotheken von einem Programm oder einer Bibliothek benötigt werden.
lddlibc4	Unterstützt **ldd** bei der Arbeit mit Objektdateien.
locale	Zeigt verschiedene Informationen über die aktuelle Locale an.
localedef	Erzeugt Locale-Spezifikationen.
mtrace	Liest und interpretiert eine Speicher-Rückverfolgungsdatei und gibt eine normal

lesbare Zusammenfassung aus.

nscd — Der „name service cache daemon"; er stellt einen Zwischenspeicher für die meisten namensbasierten Anfragen zur Verfügung.

nscd_nischeck — Prüft, ob für NIS+-Anfragen der sichere Modus benötigt wird.

pcprofiledump — Gibt Informationen aus, die durch PC-Profiling erzeugt wurden.

pt_chown — Ist ein Hilfsprogramm zu **grantpt**. Es setzt Besitzer, Gruppe und Zugriffsberechtigungen von Slave-Pseudo-Terminals.

rpcgen — Erzeugt C-Code zum Implementieren des RPC-Protokolls.

rpcinfo — Generiert eine RPC-Anfrage an einen RPC-Server.

sln — Dies ist die statisch gelinkte Variante von **ln**.

sprof — Liest Profiling-Daten zu Shared-Objects und zeigt sie an.

tzselect — Stellt dem Anwender einige Fragen zu seinem Standort und erzeugt aus den Antworten eine passende Zeitzonenbeschreibung.

xtrace — Verfolgt den Durchlauf eines Programmes, indem es die jeweils ausgeführte Funktion ausgibt.

zdump — Gibt Zeitzonen aus.

zic — Ist ein Compiler für Zeitzonen.

`ld.so` — Ist ein Hilfsprogramm für ausführbare gemeinsame Bibliotheken.

`libBrokenLocale` — Wird intern von der GLibc verwendet, um kaputte Programme (z. B. einige Motif-Programme) zum Laufen zu bekommen. Schauen Sie sich dazu die Kommentare in `glibc-2.3.6/locale/broken_cur_max.c` an.

`libSegFault` — Kümmert sich um die Verarbeitung von Speicherzugriffsfehlern; wird von **catchsegv** eingesetzt.

`libanl` — Eine Bibliothek zum asynchronen Nachschlagen von Namen.

`libbsd-compat` — Mit Hilfe dieser Bibliothek können einige BSD-Programme unter Linux lauffähig gemacht werden.

`libc` — Dies ist die C-Bibliothek.

`libcidn` — Wird intern von der Glibc zur Unterstützung von internationlisierten Domänennamen mit der Funktion `getaddrinfo()` verwendet.

`libcrypt` — Dies ist die Kryptographie-Bibliothek.

`libdl` — Eine Schnittstellenbibliothek zum dynamischen Linker.

`libg` — Eine Dummy-Bibliothek ohne jegliche Funktionen. Dies war früher eine Laufzeitbibliothek für **g++**.

`libieee` — Das Einbinden (verlinken) dieses Moduls erzwingt die Regeln der IEEE (Institute of Electrical and Electronic Engineers) zur Fehlerbehandlung mathematischer Funktionen. Standard sind die POSIX.1-Regeln zur Fehlerbehandlung.

`libm` — Die mathematische Bibliothek.

`libmcheck` — Das Einbinden (verlinken) dieses Moduls schaltet Prüfungen der

	Speicherzuordnungen ein.
`libmemusage`	Wird von **memusage** verwendet und hilft beim Sammeln von Informationen über die Speichernutzung eines Programms.
`libnsl`	Dies ist die Bibliothek für Netzwerkdienste.
`libnss`	Die Name Service Switch-Bibliotheken. Sie enthalten Funktionen zum Auflösen von Hostnamen, Benutzernamen, Gruppennamen, Aliasen, Diensten, Protokollen und so weiter.
`libpcprofile`	Enthält Profiling-Funktionen, die zum Verfolgen der CPU-Benutzung einzelner Quelltextzeilen verwendet werden können.
`libpthread`	Die POSIX-Threads-Bibliothek.
`libresolv`	Enthält Funktionen zum Erzeugen, Senden und Auswerten von Paketen an Internet Domain Name Server (DNS).
`librpcsvc`	Enthält Funktionen, die verschiedene RPC-Dienste zur Verfügung stellen.
`librt`	Diese Bibliothek enthält Funktionen mit Schnittstellen für die meisten POSIX.1b Echtzeiterweiterungen.
`libthread_db`	Enthält Funktionen, die zum Erzeugen von Debuggern für Multi-Thread-Programme nützlich sind.
`libutil`	Enthält Code für „Standard“-Funktionen, die in vielen verschiedenen Unix-Werkzeugen genutzt werden.

6.10. Erneutes Anpassen der Toolchain

Nachdem die neue C-Bibliothek nun installiert ist, muss die Toolchain erneut angepasst werden. Modifizieren Sie sie so, dass alle weiteren kompilierten Programme gegen die neue C-Bibliothek gelinkt werden. Im Grunde ist das fast das gleiche, was Sie im vorigen Kapitel beim Anpassen der Glibc schonmal gemacht haben, auch wenn es aussieht, als wäre es genau umgekehrt: Im vorigen Kapitel haben Sie die Toolchain von `/{,usr/}lib` auf dem Host in den neuen Ordner `/tools/lib` umgelenkt. Nun lenken Sie die Toolchain von diesem Ordner `/tools/lib` um auf unsere LFS-Ordner `/{,usr/}lib`.

Erstellen Sie zunächst eine Sicherungskopie des Linkers in `/tools` und ersetzen Sie ihn dann mit dem angepassten Linker aus Kapitel 5. Zu seinem Gegenstück in `/tools/$(gcc -dumpmachine)/bin` werden wir ebenfalls eine symbolische Verknüpfung einrichten.

```
mv -v /tools/bin/{ld,ld-old}
mv -v /tools/$(gcc -dumpmachine)/bin/{ld,ld-old}
mv -v /tools/bin/{ld-new,ld}
ln -sv /tools/bin/ld /tools/$(gcc -dumpmachine)/bin/ld
```

Als nächstes müssen Sie GCCs specs-Datei so bearbeiten, dass sie den neuen dynamischen Linker referenziert damit GCC seine Startdateien findet. Diese Aufgabe wird von einem einfachen **perl**-Kommando erledigt:

```
gcc -dumpspecs | \
perl -p -e 's@/tools/lib/ld-linux.so.2@/lib/ld-linux.so.2@g;' \
    -e 's@\*startfile_prefix_spec:\n@$_/usr/lib/ @g;' > \
    `dirname $(gcc --print-libgcc-file-name)`/specs
```

Danach sollten Sie die specs-Datei überprüfen und sicherstellen, dass alle gewünschten Änderungen wirklich durchgeführt wurden.



Wichtig

Wenn Sie mit einer Plattform arbeiten, bei der der Name des Linkers nicht `ld-linux.so.2` ist, *müssen* Sie in den obigen Kommandos „ld-linux.so.2“ durch dem Namen des Linkers für Ihre Plattform ersetzen. Wenn nötig, schlagen Sie nochmal im Abschnitt Abschnitt 5.2, „Technische Anmerkungen zur Toolchain,“ nach.

An dieser Stelle ist es zwingend nötig, die grundlegenden Funktionen (Kompilieren und Linken) der angepassten Toolchain zu überprüfen. Aus diesem Grund führen Sie bitte die folgenden Tests durch:

```
echo 'main(){}' > dummy.c
cc dummy.c -Wl,--verbose &> dummy.log
readelf -l a.out | grep ': /lib'
```

Wenn alles korrekt funktioniert, sollten keine Fehler auftreten und die Ausgabe des letzten Kommandos ist:

```
[Requesting program interpreter: /lib/ld-linux.so.2]
```

Beachten Sie, dass nun `/lib` der Prefix zum dynamischen Linker ist.

Überprüfen Sie nun, dass die korrekten Startdateien verwendet werden:

```
grep -o '/usr/lib.*/crt[1in].* .*' dummy.log
```

Wenn alles korrekt funktioniert, sollten keine Fehler auftreten und die Ausgabe des letzten Kommandos sieht so oder so ähnlich aus:

```
/usr/lib/crt1.o succeeded
/usr/lib/crti.o succeeded
/usr/lib/crtn.o succeeded
```

Stellen Sie als nächstes sicher, dass der neue Linker mit den korrekten Suchpfaden verwendet wird:

```
grep 'SEARCH.*/usr/lib' dummy.log |sed 's|; |\n|g'
```

Wenn alles korrekt funktioniert, sollten keine Fehler auftreten und die Ausgabe des letzten Kommandos sieht so oder so ähnlich aus:

```
SEARCH_DIR("/tools/i686-pc-linux-gnu/lib")
SEARCH_DIR("/usr/lib")
SEARCH_DIR("/lib");
```

Danach prüfen Sie, ob die korrekte libc eingesetzt wird:

```
grep "/lib/libc.so.6 " dummy.log
```

Wenn alles korrekt funktioniert, sollten keine Fehler auftreten und die Ausgabe des letzten Kommandos sieht so oder so ähnlich aus:

```
attempt to open /lib/libc.so.6 succeeded
```

Und zum Schluss kontrollieren Sie noch, ob GCC den richtigen dynamischen Linker benutzt:

```
grep found dummy.log
```

Wenn alles korrekt funktioniert, sollten keine Fehler auftreten und die Ausgabe des letzten Kommandos ist:

```
found ld-linux.so.2 at /lib/ld-linux.so.2
```

Wenn Sie eine andere oder überhaupt keine Ausgabe erhalten, ist etwas ernsthaft schiefgelaufen. Sie müssen das überprüfen und alle bisherigen Schritte noch einmal nachvollziehen, um das Problem zu finden und zu beheben. Machen Sie nicht weiter, solange das Problem nicht behoben ist. Am wahrscheinlichsten ist, dass etwas beim Anpassen der specs-Datei weiter oben nicht funktioniert hat.

Wenn Sie mit dem Ergebnis zufrieden sind, löschen Sie die Testdateien:

```
rm -v dummy.c a.out dummy.log
```

6.11. Binutils-2.16.1

Binutils ist eine Sammlung von Software-Entwicklungswerkzeugen. Dazu gehören zum Beispiel Linker, Assembler und weitere Programme für die Arbeit mit Objektdateien.

Geschätzte Kompilierzeit: 1.5 SBU inkl. Testsuite
Ungefähr benötigter Festplattenplatz: 172 MB inkl. Testsuite

6.11.1. Installation von Binutils

Jetzt ist ein guter Zeitpunkt um sicherzustellen, dass die Pseudo-Terminals (PTYs) in Ihrer chroot-Umgebung funktionieren. Mit dem folgenden schnellen Test sehen Sie, ob alles korrekt eingerichtet ist:

```
expect -c "spawn ls"
```

Falls die folgende Meldung erscheint, ist Ihre chroot-Umgebung nicht für PTYs vorbereitet:

```
The system has no more ptys.
Ask your system administrator to create more.
```

Das Problem muss behoben werden, bevor Sie die Testsuites von Binutils und GCC laufen lassen.

Die Dokumentation zu Binutils empfiehlt, Binutils außerhalb des Quellordners zu kompilieren:

```
mkdir -v ../binutils-build
cd ../binutils-build
```

Bereiten Sie Binutils zum Kompilieren vor:

```
../binutils-2.16.1/configure --prefix=/usr \
    --enable-shared
```

Kompilieren Sie das Paket:

```
make tooldir=/usr
```

Die Bedeutung des make-Parameters:

`tooldir=/usr`

Normalerweise ist *tooldir* (der Ordner, in den die ausführbaren Dateien endgültig installiert werden) auf `$(exec_prefix)/$(target_alias)` eingestellt. Ein i686-Computer löst dies zum Beispiel zu `/usr/i686-pc-linux-gnu` auf. Da wir aber nur für unser eigenes System installieren, brauchen wir diesen speziellen Ordner in `/usr` nicht. Diese Konfiguration fände z. B. dann Verwendung, wenn das System zum Querkompilieren genutzt würde (zum Beispiel, um auf einer Intel-Maschine Code zu generieren, der auf einem PowerPC ausgeführt werden kann).



Wichtig

In diesem Abschnitt wird die Testsuite von Binutils als *kritisch* eingestuft. Wir raten Ihnen, die Tests unter keinen Umständen zu überspringen.

Testen Sie das Ergebnis:

```
make check
```

Installieren Sie das Paket:

```
make tooldir=/usr install
```

Installieren Sie die Header-Datei `libiberty`, sie wird von einigen Paketen benötigt:

```
cp -v ../binutils-2.16.1/include/libiberty.h /usr/include
```

6.11.2. Inhalt von Binutils

Installierte Programme: addr2line, ar, as, c++filt, gprof, ld, nm, objcopy, objdump, ranlib, readelf, size, strings und strip
Installierte Bibliotheken: libiberty.a, libbfd.{a,so} und libopcodes.{a,so}

Kurze Beschreibungen

addr2line Konvertiert Programmadressen zu Dateinamen und Zeilennummern. Mit Hilfe des Programmnamens und einer Speicheradresse benutzt das Programm Debugging-Informationen in der ausführbaren Datei, um herauszufinden, welche Quelldatei und Zeilennummer mit der Adresse assoziiert ist.

ar Wird zum Erzeugen und Extrahieren von Dateien aus einem Archiv verwendet.

as Ein Assembler. Er assembliert die Ausgabe von **gcc** zu Objektdateien.

c++filt Wird vom dynamischen Linker benutzt, um C++- und Java-Symbole aufzuschlüsseln, damit überladene Funktionen nicht in Konflikt geraten.

gprof Zeigt call graph-Profiling-Daten an.

ld Ein Linker. Er verbindet mehrere Objektdateien und Archivdateien zu einer einzigen Datei, replaziert ihre Daten und verbindet ihre Symbolreferenzen.

nm Listet alle in einer Objektdatei vorkommenden Symbole auf.

objcopy Wird zum Konvertieren eines bestimmten Objektdateityps in einen anderen verwendet.

objdump Zeigt ausgewählte Informationen über eine Objektdatei an. Diese Informationen sind hauptsächlich für Programmierer sinnvoll, die an den Kompilierwerkzeugen arbeiten.

ranlib Erzeugt einen Index des Archivinhalts und speichert ihn im Archiv. Der Index listet alle reallokierbaren Symbole auf, die von im Archiv enthaltenen Objektdateien definiert werden.

readelf Zeigt Informationen über Binärdateien vom Typ ELF an.

size Listet die Abschnitts- und Gesamtgröße für eine Objektdatei auf.

strings Gibt für jede angegebene Datei die druckbaren Zeichenketten aus, die eine festgelegte

	Mindestgröße haben (Voreinstellung ist 4). Bei Objektdateien gibt es in der Voreinstellung nur die Zeichenketten aus den Initialisierungs- und Ladeabschnitten aus. Bei anderen Dateitypen durchsucht es die gesamte Datei.
strip	Entfernt bestimmte Symbole aus Objektdateien (z. B. Debugging-Symbole).
`libiberty`	Enthält Routinen, die von verschiedenen GNU-Programmen genutzt werden, inklusive **getopt**, **obstack**, **strerror**, **strtol** und **strtoul**.
`libbfd`	Die Bibliothek für Binärdateibezeichner.
`libopcodes`	Eine Bibliothek zur Behandlung von Obcodes. Sie wird zum Erzeugen von Werkzeugen wie z. B. **objdump** benutzt. Obcodes sind die „lesbaren“ Versionen der Prozessorinstruktionen.

6.12. GCC-4.0.3

Das Paket GCC enthält die GNU-Compiler-Sammlung. Darin sind die C- und C++-Compiler enthalten.

Geschätzte Kompilierzeit: 22 SBU inkl. Testsuite
Ungefähr benötigter Festplattenplatz: 566 MB inkl. Testsuite

6.12.1. Installation von GCC

Wenden Sie nun einen **Sed**-Befehl an um die Installation von `libiberty.a` zu verhindern. Wir möchten die von Binutils bereitgestellte Version von `libiberty.a` verwenden:

```
sed -i 's/install_to_$(INSTALL_DEST) //' libiberty/Makefile.in
```

Im Bootstrap-Durchlauf aus Abschnitt 5.4, „GCC-4.0.3 - Durchlauf 1“ wurde zum Kompilieren von GCC der Compiler-Parameter `-fomit-frame-pointer` verwendet. Der Nicht-Bootstrap-Durchlauf verwendet diesen Parameter jedoch standardmäßig nicht. Um die Kompilier-Durchläufe von GCC konsistent zu halten, sollten Sie den Parameter für diesen Durchlauf mit dem folgenden **sed**-Kommando einschalten.

```
sed -i 's/^XCFLAGS =$/& -fomit-frame-pointer/' gcc/Makefile.in
```

Das Skript **fixincludes** versucht manchmal, die bereits installieren Header-Dateien des Systems zu "reparieren". Es ist uns allerdings bekannt, dass weder die Header von GCC-4.0.3 noch die von Glibc-2.3.6 eine Reparatur benötigen. Daher verhindern Sie den Start des **fixincludes**-Skriptes mit diesem Kommando:

```
sed -i 's@\./fixinc\.sh@-c true@' gcc/Makefile.in
```

GCC enthält ein Skript namens **gccbug**, welches zum Kompilierzeitpunkt feststellt, ob mktemp vorhanden ist. Das Ergebnis wird fest in einen bestimmten Test eingebunden. Das wiederum würde den Test dazu veranlassen, weniger sichere Zufallsnamen für temporäre Dateien zu erzeugen. Da wir mktemp später noch installieren werden, simulieren wir an dieser Stelle das Vorhandensein.

```
sed -i 's/@have_mktemp_command@/yes/' gcc/gccbug.in
```

Die Dokumentation zu GCC empfiehlt, GCC außerhalb des Quellordners zu kompilieren:

```
mkdir -v ../gcc-build
cd ../gcc-build
```

Bereiten Sie GCC zum Kompilieren vor:

```
../gcc-4.0.3/configure --prefix=/usr \
    --libexecdir=/usr/lib --enable-shared \
    --enable-threads=posix --enable-__cxa_atexit \
    --enable-clocale=gnu --enable-languages=c,c++
```

Kompilieren Sie das Paket:

```
make
```



Wichtig

In diesem Abschnitt wird die Testsuite als *absolut kritisch* betrachtet. Sie sollten diesen Schritt unter keinen Umständen überspringen.

Testen Sie die Ergebnisse, aber halten Sie bei Fehlern nicht an:

```
make -k check
```

Um eine Zusammenfassung der Testergebnisse zu sehen, verwenden Sie dieses Kommando:

```
../gcc-4.0.3/contrib/test_summary
```

Wenn Sie nur die Zusammenfassungen sehen möchten, pipen Sie die Ausgabe durch **grep -A7 Summ**.

Sie können die Ergebnisse mit denen unter *http://www.linuxfromscratch.org/lfs/build-logs/6.2/* vergleichen.

Ein paar unerwartete Fehler lassen sich oftmals nicht vermeiden. Die Entwickler von GCC kennen diese üblicherweise bereits, hatten aber noch keine Zeit, diese Fehler zu beheben. Insbesondere die `libmudflap`-Tests sind aufgrund eines Fehlers ins GCC anfällig (*http://gcc.gnu.org/bugzilla/show_bug.cgi?id=20003*).Kurz gesagt, solange Ihre Testergebnisse nicht grob von denen unter der obigen URL abweichen, können Sie beruhigt fortfahren.

Installieren Sie das Paket:

```
make install
```

Einige Pakete erwarten, dass der C-Präprozessor unter `/lib` installiert ist. Erstellen Sie daher diesen symbolischen Link:

```
ln -sv ../usr/bin/cpp /lib
```

Viele Pakete benutzen den Namen **cc**, um den C-Compiler aufzurufen. Um auch diesen Paketen Rechnung zu tragen, erzeugen Sie einen weiteren symbolischen Link:

```
ln -sv gcc /usr/bin/cc
```

Die endgültige Toolchain ist nun fertiggestellt. An dieser Stelle muss unbedingt erneut überprüft werden, ob Kompilieren und Linken mit ihr wie erwartet funktioniert. Wir führen den gleichen Gesundheitstest wie schon einmal in diesem Kapitel durch:

```
echo 'main(){}' > dummy.c
cc dummy.c -Wl,--verbose &> dummy.log
readelf -l a.out | grep ': /lib'
```

Wenn alles korrekt funktioniert, sollten keine Fehler auftreten und die Ausgabe des letzten Kommandos ist:

```
[Requesting program interpreter: /lib/ld-linux.so.2]
```

Überprüfen Sie nun, dass die korrekten Startdateien verwendet werden:

```
grep -o '/usr/lib.*/crt[1in].* .*' dummy.log
```

Wenn alles korrekt funktioniert, sollten keine Fehler auftreten und die Ausgabe des letzten Kommandos sieht so oder so ähnlich aus:

```
/usr/lib/gcc/i686-pc-linux-gnu/4.0.3/../../../crt1.o succeeded
/usr/lib/gcc/i686-pc-linux-gnu/4.0.3/../../../crti.o succeeded
/usr/lib/gcc/i686-pc-linux-gnu/4.0.3/../../../crtn.o succeeded
```

Stellen Sie als nächstes sicher, dass der neue Linker mit den korrekten Suchpfaden verwendet wird:

```
grep 'SEARCH.*/usr/lib' dummy.log |sed 's|; |\n|g'
```

Wenn alles korrekt funktioniert, sollten keine Fehler auftreten und die Ausgabe des letzten Kommandos sieht so oder so ähnlich aus:

```
SEARCH_DIR("/usr/i686-pc-linux-gnu/lib")
SEARCH_DIR("/usr/local/lib")
SEARCH_DIR("/lib")
SEARCH_DIR("/usr/lib");
```

Danach prüfen Sie, ob die korrekte libc eingesetzt wird:

```
grep "/lib/libc.so.6 " dummy.log
```

Wenn alles korrekt funktioniert, sollten keine Fehler auftreten und die Ausgabe des letzten Kommandos sieht so oder so ähnlich aus:

```
attempt to open /lib/libc.so.6 succeeded
```

Und zum Schluss kontrollieren Sie noch, ob GCC den richtigen dynamischen Linker benutzt:

```
grep found dummy.log
```

Wenn alles korrekt funktioniert, sollten keine Fehler auftreten und die Ausgabe des letzten Kommandos ist:

```
found ld-linux.so.2 at /lib/ld-linux.so.2
```

Wenn Sie eine andere oder überhaupt keine Ausgabe erhalten, ist etwas ernsthaft schiefgelaufen. Sie müssen das überprüfen und alle bisherigen Schritte noch einmal nachvollziehen, um das Problem zu finden und zu beheben. Machen Sie nicht weiter, solange das Problem nicht behoben ist. Am wahrscheinlichsten ist, dass etwas beim Anpassen der specs-Datei weiter oben nicht funktioniert hat.

Wenn Sie mit dem Ergebnis zufrieden sind, löschen Sie die Testdateien:

```
rm -v dummy.c a.out dummy.log
```

6.12.2. Inhalt von GCC

Installierte Programme: c++, cc (Link auf gcc), cpp, g++, gcc, gccbug und gcov
Installierte Bibliotheken: libgcc.a, libgcc_eh.a, libgcc_s.so, libstdc++.{a,so} und libsupc++.a

Kurze Beschreibungen

cc	Dies ist der C-Compiler.
cpp	Ein C-Präprozessor. Er wird vom Compiler benutzt, um #include, #define und ähnliche Anweisungen im Quellcode durch ihren endgültigen Code zu ersetzen.
c++	Dies ist der C++-Compiler.
g++	Dies ist der C++-Compiler.
gcc	Dies ist der C-Compiler.
gccbug	Ein Shellskript, mit dem man nützliche Fehlerberichte erzeugen kann.
gcov	Ein Werkzeug zum Testen des Deckungsgrades. Es wird zum Analysieren von Programmen benutzt, um herauszufinden, wo Optimierungen den größten Effekt zeigen.
`libgcc`	Enthält Laufzeitunterstützung für **gcc**.
`libstdc++`	Die Standard-C++-Bibliothek.
`libsupc++`	Stellt Unterstützungsroutinen für die Programmiersprache C++ zur Verfügung.

6.13. Berkeley DB-4.4.20

Das Paket Berkeley DB enthält Programme und Werkzeuge, die von vielen Anwendungen für Datenbankbezogene Funktionen verwendet werden.

Geschätzte Kompilierzeit: 1.2 SBU
Ungefähr benötigter Festplattenplatz: 77 MB



Weitere Installationsmöglichkeiten

Das BLFS-Buch enthält eine Anleitung zur Installation dieses Pakets, falls Sie einen RPC-Server oder andere Sprachbindungen benötigen. Die zusätzlichen Sprachbindungen setzen weitere Pakete voraus. Weitere Informationen dazu finden Sie unter *http://www.linuxfromscratch.org/blfs/view/svn/server/databases.html#db*.

Außerdem *kann* man anstelle von Berkeley DB auch GDBM installieren und somit die Voraussetzung für Man-DB schaffen. Allerdings sind viele Stunden in den LFS-Test von Berkeley geflossen, nicht jedoch in GDBM. Wenn Sie sich dem Risiko voll bewusst sind, und dennoch GDBM einsetzen möchten, dann schauen Sie sich die Anleitungen unter *http://www.linuxfromscratch.org/blfs/view/svn/general/gdbm.html* an.

6.13.1. Installation von Berkeley DB

Patchen Sie das Paket, um ein paar mögliche Fallstricke zu umgehen:

```
patch -Np1 -i ../db-4.4.20-fixes-1.patch
```

Bereiten Sie Berkeley DB zum Kompilieren vor:

```
cd build_unix &&
../dist/configure --prefix=/usr --enable-compat185 --enable-cxx
```

Die Bedeutung der configure-Parameter:

--enable-compat185
: Dieser Parameter schaltet die Berkeley DB 1.85 Kompatibilitäts-API ein.

--enable-cxx
: Dieser Parameter schaltet den Bau der C++-API-Bibliotheken ein.

Kompilieren Sie das Paket:

```
make
```

Es ist nicht möglich, dieses Paket sinnvoll zu testen, weil dafür die TCL-Bindungen voraussetzt. Die TCL-Bindungen können allerdings nicht korrekt kompiliert werden, weil TCL gegen die Glibc in `/tools` gelinkt ist und nicht die in `/usr`.

Installieren Sie das Paket:

```
make docdir=/usr/share/doc/db-4.4.20 install
```

Die Bedeutung des make-Parameters:

docdir=...
Diese Variable gibt den korrekten Speicherort für die Dokumentation an.

Korrigieren Sie den Besitzer der installierten Dateien:

```
chown -v root:root /usr/bin/db_* \
    /usr/lib/libdb* /usr/include/db* &&
chown -Rv root:root /usr/share/doc/db-4.4.20
```

6.13.2. Inhalt von Berkeley DB

Installierte Programme: db_archive, db_checkpoint, db_deadlock, db_dump, db_hotbackup, db_load, db_printlog, db_recover, db_stat, db_upgrade und db_verify
Installierte Bibliotheken: libdb.{so,ar} und libdb_cxx.r{o,ar}

Kurze Beschreibungen

db_archive	Gibt die Pfade zu Protokolldateien aus, die nicht mehr benutzt werden.
db_checkpoint	Ein Daemon zum Überwachen von Protokolldateien und Kontrollpunkten darin.
db_deadlock	Ein Daemon zum Unterbrechen von Sperrungen, falls eine ununterbrechbare Sperrung (deadlock) gefunden wird.
db_dump	Wandelt eine Datenbankdatei in eine reines Textdatei um, so dass sie von **db_load** gelesen werden kann.
db_hotbackup	Erzeugt Schnappschüsse einer Berkeley DB Datenbank zum Zweck eines „Online-Backup“ oder „Online-Failover“.
db_load	Wird zum Erzeugen einer Datenbank-Datei aus einer reinen Text-Datei verwendet.
db_printlog	Wandelt eine Protokolldatei einer Datenbank in ein von Menschen lesbares Format um.
db_recover	Stellt eine Datenbank nach einem Fehler wieder in einem konsistenten Zustand her.
db_stat	Zeigt Statistiken zu Berkeley Datenbanken an.
db_upgrade	Wird zum Aktualisieren von Datenbank-Dateien auf eine neuere Berkeley DB-Version verwendet.
db_verify	Wird zum Durchführen von Konsistenzprüfungen von Datenbank-Dateien verwendet.
`libdb.{so,a}`	Enthält Funktionen zum Manipulieren von Datenbank-Dateien aus C-Programmen heraus.
`libdb_cxx.{so,a}`	Enthält Funktionen zum Manipulieren von Datenbank-Dateien aus C++-Programmen heraus.

6.14. Coreutils-5.96

Das Paket Coreutils enthält viele Shell-Werkzeuge zum Einstellen der grundlegenden Systemeigenschaften.

Geschätzte Kompilierzeit: 1.1 SBU
Ungefähr benötigter Festplattenplatz: 58.3 MB

6.14.1. Installation von Coreutils

Die Funktion von **uname** ist bekannterweise ein wenig fehlerhaft, weil der Parameter *-p* immer `unknown` ausgibt. Der folgende Patch behebt das Problem auf Intel-Architekturen:

```
patch -Np1 -i ../coreutils-5.96-uname-1.patch
```

Normalerweise würde Coreutils einige Programme installieren, die später von anderen Paketen bereitgestellt werden sollen. Verhinden Sie die Installation dieser Programme mit diesem Patch:

```
patch -Np1 -i ../coreutils-5.96-suppress_uptime_kill_su-1.patch
```

Von POSIX wird verlangt, dass die Programme von Coreutils Zeichengrenzen auch in Multibyte-Locales erkennen. Der folgende Patch behebt einige diesbezügliche Fehler:

```
patch -Np1 -i ../coreutils-5.96-i18n-1.patch
```

Damit die vom Patch hinzugefügten Tests erfolgreich durchlaufen können, müssen die Zugriffsrechte für die Testdatei geändert werden:

```
chmod +x tests/sort/sort-mb-tests
```



Anmerkung

In der Vergangenheit wurden leider viele Fehler in diesem Patch gefunden. Wenn Sie neue Fehler an die Entwickler von Coreutils berichten möchten, prüfen Sie bitte zuallererst, ob sich der Fehler auch ohne diesen Patch noch reproduzieren lässt!

Es hat sich herausgestellt, dass übersetzte Texte manchmal einen Puffer in **who -Hu** überlaufen lassen. Erhöhen Sie die Puffergröße:

```
sed -i 's/_LEN 6/_LEN 20/' src/who.c
```

Bereiten Sie Coreutils zum Kompilieren vor:

```
./configure --prefix=/usr
```

Kompilieren Sie das Paket:

```
make
```

Diese Testsuite benötigt einige System-Benutzer und -Gruppen um korrekt zu funktionieren. Wenn Sie die Testsuite laufen lassen möchten, müssen Sie vorher die folgenden Befehle ausführen. Falls Sie die Testsuite nicht ausführen möchten überspringen Sie die Befehle und machen einfach bei „Installieren Sie das Paket“ fort.

Erstellen Sie zwei Dummy-Gruppen und einen Dummy-Benutzer:

```
echo "dummy1:x:1000:" >> /etc/group
echo "dummy2:x:1001:dummy" >> /etc/group
echo "dummy:x:1000:1000::/root:/bin/bash" >> /etc/passwd
```

Sie können die Testsuite nun durchlaufen lassen. Als erstes starten Sie einige Tests, die als `root` laufen müssen:

```
make NON_ROOT_USERNAME=dummy check-root
```

Die verbleibenden Tests werden als Benutzer `dummy` ausgeführt:

```
src/su dummy -c "make RUN_EXPENSIVE_TESTS=yes check"
```

Danach entfernen Sie die dummy Gruppen und Benutzer:

```
sed -i '/dummy/d' /etc/passwd /etc/group
```

Installieren Sie das Paket:

```
make install
```

Und verschieben Sie einige Programme an die von FHS vorgegebene Stelle:

```
mv -v /usr/bin/{cat,chgrp,chmod,chown,cp,date,dd,df,echo} /bin
mv -v /usr/bin/{false,hostname,ln,ls,mkdir,mknod,mv,pwd,rm} /bin
mv -v /usr/bin/{rmdir,stty,sync,true,uname} /bin
mv -v /usr/bin/chroot /usr/sbin
```

Einige der LFS-Bootskripte sing abhängig von den Kommandos **head** und **sleep**. Da `/usr` in den früheren Phasen des Bootvorgangs noch nicht einghängt sein könnte, müssen sich diese Programm auf der root-Partition befinden:

```
mv -v /usr/bin/{head,sleep,nice} /bin
```

6.14.2. Inhalt von Coreutils

Installierte Programme: basename, cat, chgrp, chmod, chown, chroot, cksum, comm, cp, csplit, cut, date, dd, df, dir, dircolors, dirname, du, echo, env, expand, expr, factor, false, fmt, fold, groups, head, hostid, hostname, id, install, join, link, ln, logname, ls, md5sum, mkdir, mkfifo, mknod, mv, nice, nl, nohup, od, paste, pathchk, pinky, pr, printenv, printf, ptx, pwd, readlink, rm, rmdir, seq, sha1sum, shred, sleep, sort, split, stat, stty, sum, sync, tac, tail, tee, test, touch, tr, true, tsort, tty, uname, unexpand, uniq, unlink, users, vdir, wc, who, whoami und yes

Kurze Beschreibungen

basename	Entfernt den Pfad und Suffix von einem angegebenen Dateinamen.
cat	Gibt Dateien an der Standardausgabe aus bzw. fügt sie zusammen.
chgrp	Ändert die Gruppenzugehörigkeit von Dateien und Ordnern.
chmod	Ändert die Zugriffsrechte der angegebenen Dateien. Der Modus kann entweder symbolisch (in Form der durchzuführenden Änderungen) oder als Oktalzahl angegeben werden (repräsentiert die absoluten neuen Rechte).

chown Ändert Besitzer und/oder Gruppenzugehörigkeit der angegebenen Dateien und Ordner.

chroot Macht den angegebenen Ordner temporär zum neuen Basisordner („/“) für den übergebenen Befehl (z. B. **bash**). Der Befehl wird dann in diesem „Gefängnis“ ausgeführt.

cksum Gibt die CRC-Prüfsumme (Cyclic Redundancy Check) und die Anzahl der Bytes einer angegebenen Datei aus.

comm Vergleicht zwei sortierte Dateien und gibt in drei Spalten die Zeilen aus, die jeweils einzigartig bzw. gleich sind.

cp Kopiert Dateien.

csplit Teilt eine Datei in mehrere neue Dateien. Dazu wird ein bestimmtes Muster oder Zeilennummern verwendet. Außerdem gibt **csplit** die Anzahl Bytes der neuen Dateien aus.

cut Gibt Ausschnitte von Zeilen aus. Die Ausschnitte werden nach Feldern oder Positionsangaben gewählt.

date Gibt die aktuelle Zeit im angegebenen Format aus oder stellt die Systemzeit ein.

dd Kopiert eine Datei mit der angegebenen Blockgröße und -anzahl. Optional kann währenddessen eine Konvertierung durchgeführt werden.

df Berichtet über den verfügbaren (und verwendeten) Festplattenspeicher auf allen eingehängten Dateisystemen oder den Dateisystemen, die die angegebenen Dateien enthalten.

dir Listet den Inhalt eines Ordners auf (das Gleiche wie **ls**).

dircolors Gibt Kommandos zum Setzen der Umgebungsvariable `LS_COLOR` aus, um damit das Farbschema von **ls** zu ändern.

dirname Entfernt den nicht-ordnerspezifischen Teil eines Dateinamens.

du Gibt aus, wieviel Festplattenspeicher der aktuelle Ordner, die Unterordner und Dateien oder eine einzelne Datei verbraucht.

echo Gibt eine angegebene Zeichenkette aus.

env Führt ein Kommando in einer modifizierten Arbeitsumgebung aus.

expand Konvertiert Tabulatoren zu Leerzeichen.

expr Wertet einen Ausdruck aus.

factor Gibt den Primfaktor aller angegebenen Ganzzahlen aus.

false Tut gar nichts, ist immer erfolglos. Es beendet sich immer mit einem Abschlusscode, der auf einen Fehler hinweist.

fmt Formatiert die Absätze in der übergebenen Datei neu.

fold Fügt Zeilenumbrüche in den angegebenen Dateien ein.

groups Gibt die Gruppenzugehörigkeit eines Benutzers aus.

head Gibt die ersten zehn (oder die angegebene Anzahl) von Zeilen einer Datei aus.

hostid Gibt die numerische ID (hexadezimal) des Systems aus.

hostname Setzt den Hostnamen bzw. zeigt ihn an.

id Gibt die effektive Benutzer-ID, Gruppen-ID, und Gruppenzugehörigkeit des aktuellen

Benutzers oder eines angegebenen Benutzers aus.

install Kopiert Dateien und setzt deren Zugriffsrechte und, falls möglich, Besitzer und Gruppe.

join Fügt aus zwei Dateien die Zeilen mit identischen join-Feldern zusammen.

link Erzeugt einen harten Link von der angegebenen Datei zu einer Datei.

ln Erzeugt einen harten oder symbolischen Link zwischen Dateien.

logname Gibt den Login-Namen des aktuellen Benutzers aus.

ls Listet den Inhalt des angegebenen Ordners auf.

md5sum Erzeugt eine MD5-Prüfsumme (Message Digest 5) bzw. zeigt sie an.

mkdir Erzeugt Ordner mit den angegebenen Namen.

mkfifo Erzeugt FIFO's (First-In, First-Out, eine sogenannte "named Pipe" im UNIX Sprachgebrauch) mit dem angegebenen Namen.

mknod Erzeugt eine Gerätedatei mit dem angegebenen Namen. Eine Gerätedatei ist eine spezielle zeichen- oder blockorientierte Datei oder ein FIFO.

mv Verschiebt Dateien und Ordner oder benennt sie um.

nice Führt ein Programm mit geänderter Priorität aus.

nl Nummeriert die Zeilen der angegebenen Dateien.

nohup Führt ein Programm aus, so dass es immun gegen „hangup“s ist. Die Ausgaben des Programms werden in eine Protokolldatei umgeleitet.

od Gibt eine Datei oktal oder in anderen Formaten aus.

paste Fügt angegebene Dateien zusammen. Sequenziell zusammengehörende Zeilen werden Seite an Seite durch Tabulatoren getrennt zusammengefügt.

pathchk Prüft, ob Dateinamen gültig und portierbar sind.

pinky Eine abgespeckte Version von finger. Es gibt ein paar Informationen über den angegebenen Benutzer aus.

pr Bereitet Dateien seiten- oder spaltenweise für den Ausdruck vor.

printenv Gibt die Umgebungsvariablen aus.

printf Gibt die angegebenen Argumente in einem bestimmten Format aus — dies ist der C-Funktion printf sehr ähnlich.

ptx Erzeugt aus dem Inhalt von Dateien einen vertauschten Index, mit jedem Stichwort im Kontext.

pwd Gibt den Namen des aktuellen Arbeits-Ordners aus.

readlink Gibt das Ziel eines symbolischen Links aus.

rm Löscht Dateien oder Ordner.

rmdir Löscht leere Ordner.

seq Gibt eine Zahlenreihe in einem bestimmten Wertebereich und mit einem bestimmten Inkrement aus.

sha1sum Prüft 160-Bit SHA1-Prüfsummen oder gibt sie aus.

shred Überschreibt eine Datei mehrfach mit zufälligen Mustern, um das Wiederherstellen der Daten zu erschweren.

sleep Pausiert für die angegebene Zeit.

sort Sortiert die Zeilen einer Datei.

split Teilt eine Datei in Stücke, nach Größe oder nach Zeilennummern.

stat Zeigt den Datei- oder Dateisystemstatus an.

stty Setzt Terminal-Einstellungen oder zeigt sie an.

sum Gibt Prüfsumme und Anzahl der Blöcke einer Datei aus.

sync Schreibt den Dateisystempuffer. Geänderte Blöcke werden auf die Festplatte geschrieben und der Superblock wird aktualisiert.

tac Fügt Dateien rückwärts zusammen.

tail Gibt die letzten zehn (oder die angegebene Anzahl) von Zeilen einer Datei aus.

tee Liest von der Standardeingabe während gleichzeitig auf die Standardausgabe und in eine Datei geschrieben wird.

test Vergleicht Werte und prüft Dateitypen.

touch Ändert Zeitstempel von Dateien, setzt Zugriffs- und Änderungszeit einer Datei auf die aktuelle Zeit. Dateien, die noch nicht existieren, werden mit der Länge 0 angelegt.

tr Übersetzt, quetscht oder entfernt Zeichen von der Standardeingabe.

true Macht nichts, ist immer erfolgreich. Beendet immer mit einem Statuscode, der Erfolg bedeutet.

tsort Sortiert topologisch. Schreibt eine vollständig sortierte Liste entsprechend der teilweisen Sortierung in einer Datei.

tty Gibt den Dateinamen des Terminals aus, das mit der Standardeingabe verbunden ist.

uname Gibt Systeminformationen aus.

unexpand Konvertiert Leerzeichen zu Tabulatoren.

uniq Entfernt alle identischen Zeilen bis auf eine.

unlink Entfernt eine Datei.

users Gibt die Namen der eingeloggten Benutzer aus.

vdir Macht das Gleiche wie **ls -l**.

wc Gibt die Anzahl Zeilen, Wörter und Bytes einer Datei aus. Und eine Summe, falls mehrere Dateien angegeben wurden.

who Zeigt an, wer gerade eingeloggt ist.

whoami Gibt den Benutzernamen aus, der mit der aktuell effektiven Benutzer-ID verknüpft ist.

yes Gibt „y“ oder eine andere Zeichenkette solange aus, bis es beendet wird.

6.15. Iana-Etc-2.10

Das Paket Iana-Etc enthält Daten zu Netzwerkdiensten und Protokollen.

Geschätzte Kompilierzeit: weniger als 0.1 SBU
Ungefähr benötigter Festplattenplatz: 2.1 MB

6.15.1. Installation von Iana-Etc

Das folgende Kommando konvertiert die von IANA bereitgestellten RAW-Daten in das korrekte Format für `/etc/protocols` und `/etc/services`:

```
make
```

Dieses Paket enthält keine Testsuite.

Installieren Sie das Paket:

```
make install
```

6.15.2. Inhalt von Iana-Etc:

Installierte Dateien: /etc/protocols und /etc/services

Kurze Beschreibungen

`/etc/protocols`	Beschreibt verschiedene im TCP/IP-Subsystem verfügbare DARPA Internet-Protokolle.
`/etc/services`	Ermöglicht eine Zuordnung von leicht zu lesenden Namen für Internetdienste und den zugehörigen Port-Nummern und Protokolltypen.

6.16. M4-1.4.4

M4 enthält einen Makroprozessor.

Geschätzte Kompilierzeit: weniger als 0.1 SBU
Ungefähr benötigter Festplattenplatz: 3 MB

6.16.1. Installation von M4

Bereiten Sie M4 zum Kompilieren vor:

```
./configure --prefix=/usr
```

Kompilieren Sie das Paket:

```
make
```

Um das Ergebnis zu prüfen, führen Sie **make check** aus.

Installieren Sie das Paket:

```
make install
```

6.16.2. Inhalt von M4

Installiertes Programm: m4

Kurze Beschreibungen

m4 Kopiert die Eingabe zur Ausgabe und führt dabei Makros aus. Die Makros können entweder vordefiniert oder selbstgeschrieben sein und beliebige Argumente übernehmen. Neben der Fähigkeit, Makros auszuführen, besitzt **m4** eingebaute Funktionen zum Einfügen benannter Dateien, zum Ausführen von Unix-Befehlen und Integer-Berechnungen, zur Manipulation von Text und zur Behandlung von Rekursionen usw. **m4** kann entweder als Frontend zu einem Compiler oder als eigenständiger Makroprozessor genutzt werden.

6.17. Bison-2.2

Mit Bison lassen sich Programme generieren, die die Struktur einer Textdatei analysieren.

Geschätzte Kompilierzeit: 0.6 SBU
Ungefähr benötigter Festplattenplatz: 11.9 MB

6.17.1. Installation von Bison

Bereiten Sie Bison zum Kompilieren vor:

```
./configure --prefix=/usr
```

Das configure-System bereitet Bison ohne Unterstützung für internationalisierte Fehlermeldungen vor, wenn das Programm **bison** nicht bereits in $PATH gefunden wird. Durch den folgenden Zusatz wird dies korrigiert.

```
echo '#define YYENABLE_NLS 1' >> config.h
```

Kompilieren Sie das Paket:

```
make
```

Um das Ergebnis zu prüfen, führen Sie **`make check`** aus.

Installieren Sie das Paket:

```
make install
```

6.17.2. Inhalt von Bison

Installierte Programme: bison und yacc
Installierte Bibliothek: liby.a

Kurze Beschreibungen

bison Erzeugt aus einer Reihe von Regeln ein Programm zum Analysieren der Struktur von Textdateien. Bison ist ein Ersatz für yacc (Yet Another Compiler Compiler).

yacc Ein Wrapper zu **bison**. Er wird benutzt, weil immer noch viele Programm **yacc** anstelle von **bison** aufrufen. **Bison** wird dann mit der Option *`-y`* aufgerufen.

`liby.a` Die Yacc-Bibliothek, die die Implementierung von yacc-kompatiblen *yyerror*- und *main*-Funktionen enthält. Diese Bibliothek ist normalerweise nicht sehr nützlich, aber sie wird von POSIX vorausgesetzt.

6.18. Ncurses-5.5

Das Paket Ncurses enthält Bibliotheken für den Terminal-unabhängigen Zugriff auf Textbildschirme.

Geschätzte Kompilierzeit: 0.7 SBU
Ungefähr benötigter Festplattenplatz: 31 MB

6.18.1. Installation von Ncurses

Seit der Veröffentlichung von Ncurses-5.5 wurde ein Speicherleck und einige Anzeigefehler entdeckt und behoben. Wenden Sie diese Fehlerbereinigungen nun an:

```
patch -Np1 -i ../ncurses-5.5-fixes-1.patch
```

Bereiten Sie Ncurses zum Kompilieren vor:

```
./configure --prefix=/usr --with-shared --without-debug --enable-widec
```

Die Bedeutung des configure-Parameters:

--enable-widec
: Durch diesen Parameter werden anstelle der normalen Bibliotheken (`libncurses.so.5.5`) die Versionen für Multibyte-Zeichen installiert (`libncursesw.so.5.5`). Diese Wide-Character-Bibliotheken sind sowohl mit Multibyte- als auch mit normalen 8-Bit-Locales verwendbar. Die beiden Bibliothek-Typen sind Quell- aber nicht Binär-Kompatibel.

Kompilieren Sie das Paket:

```
make
```

Dieses Paket enthält keine Testsuite.

Installieren Sie das Paket:

```
make install
```

Vergeben Sie Ausführungsrechte für die Ncurses-Bibliothek:

```
chmod -v 755 /usr/lib/*.5.5
```

Korrigieren Sie eine Bibliothek, die nicht ausführbar sein sollte:

```
chmod -v 644 /usr/lib/libncurses++w.a
```

Verschieben Sie die Bibliotheken in den Ordner `/lib`, denn es wird erwartet, dass sie sich dort befinden:

```
mv -v /usr/lib/libncursesw.so.5* /lib
```

Da die Bibliotheken gerade verschoben wurden, zeigt ein symbolischer Links nun ins Leere. Erstellen Sie diesen neu:

```
ln -sfv ../../lib/libncursesw.so.5 /usr/lib/libncursesw.so
```

Viele Programme erwarten immer noch vom Linker, die nicht-Wide-Character-Bibliotheken von Ncurses aufzufinden. Mit symbolischen Links und Linker-Skripts können Sie diese Programme austricksen:

```
for lib in curses ncurses form panel menu ; do \
    rm -vf /usr/lib/lib${lib}.so ; \
    echo "INPUT(-l${lib}w)" >/usr/lib/lib${lib}.so ; \
    ln -sfv lib${lib}w.a /usr/lib/lib${lib}.a ; \
done &&
ln -sfv libncurses++w.a /usr/lib/libncurses++.a
```

Stellen Sie des Weiteren sicher, dass alte Programme, die mit `-lcurses` verlinken, immer noch kompilierbar sind:

```
echo "INPUT(-lncursesw)" >/usr/lib/libcursesw.so &&
ln -sfv libncurses.so /usr/lib/libcurses.so &&
ln -sfv libncursesw.a /usr/lib/libcursesw.a &&
ln -sfv libncurses.a /usr/lib/libcurses.a
```



Anmerkung

Die obigen Kommandos installieren keine nicht-Wide-Bibliotheken von Ncurses, weil kein aus dem Quellcode installierte Paket diese verwenden würde. Wenn Sie allerdings Binär-Programme haben, die diese Bibliotheken benötigen, so können die passenden Bibliotheken mit diesen Kommandos installiert werden:installi

```
make distclean &&
./configure --prefix=/usr --with-shared --without-normal \
  --without-debug --without-cxx-binding &&
make sources libs &&
cp -av lib/lib*.so.5* /usr/lib
```

6.18.2. Inhalt von Ncurses

Installierte Programme: captoinfo Link auf tic), clear, infocmp, infotocap (Link auf tic), reset (Link auf tset), tack, tic, toe, tput und tset
Installierte Bibliotheken: libcursesw.{a,so} (symlink und das Linker-Skript zu libncursesw.{a,so}), libformw.{a,so}, libmenuw.{a,so}, libncurses++w.a, libncursesw.{a,so}, libpanelw.{a,so} und ihre Nicht-Wide-Character Gegenstücke ohne "w" im Namen der Bibliothek.

Kurze Beschreibungen

captoinfo	Konvertiert termcap-Beschreibungen zu terminfo-Beschreibungen.
clear	Löscht den Bildschirminhalt (wenn möglich).
infocmp	Vergleicht terminfo-Beschreibungen oder gibt sie aus.
infotocap	Konvertiert terminfo-Beschreibungen zu termcap-Beschreibungen.
reset	Setzt ein Terminal auf seine Voreinstellungen zurück.
tack	Wird benutzt, um die Korrektheit eines Eintrages in der terminfo-Datenbank zu überprüfen.
tic	Der Compiler für Beschreibungen zu terminfo-Einträgen. Er übersetzt terminfo-Dateien aus dem Quellformat in das binäre Format, das von den ncurses-Bibliotheksroutinen benötigt wird. Eine terminfo-Datei enthält Informationen über die Fähigkeiten eines bestimmten Terminals.

toe Listet alle verfügbaren Terminaltypen auf und gibt zu jedem den Namen und die Beschreibung aus.

tput Macht der Shell die Werte von Terminal-abhängigen Fähigkeiten zugänglich. Es kann auch zum Zurücksetzen oder Initialisieren eines Terminals oder zum Anzeigen seines vollständigen Namens verwendet werden.

tset Kann zum Initialisieren eines Terminals verwendet werden.

`libcurses` Ein Link auf `libncurses`.

`libncurses` Enthält Funktionen zum Anzeigen von Text auf einem Terminal in vielen komplizierten Variationen. Ein gutes Beispiel ist das angezeigte Menü von **make menuconfig** des Kernels.

`libform` Enthält Funktionen zum Implementieren von Formularen.

`libmenu` Enthält Funktionen zum Implementieren von Menüs.

`libpanel` Enthält Funktionen zum Implementieren von Schaltflächen.

6.19. Procps-3.2.6

Procps enthält Programme zur Überwachung und Steuerung von Systemprozessen. Die Informationen zu den Prozessen erhält Procps aus dem Ordner /proc.

Geschätzte Kompilierzeit: 0.1 SBU
Ungefähr benötigter Festplattenplatz: 2.3 MB

6.19.1. Installation von Procps

Kompilieren Sie das Paket:

```
make
```

Dieses Paket enthält keine Testsuite.

Installieren Sie das Paket:

```
make install
```

6.19.2. Inhalt von Procps

Installierte Programme: free, kill, pgrep, pkill, pmap, ps, skill, slabtop, snice, sysctl, tload, top, uptime, vmstat, w und watch
Installierte Bibliothek: libproc.so

Kurze Beschreibungen

free Gibt die Menge an freiem und benutzten Arbeitsspeicher aus, sowohl physischem als auch Swap.

kill Sendet Signale an Prozesse.

pgrep Findet Prozesse aufgrund ihres Namens und anderer Attribute.

pkill Signalisiert Prozesse basierend auf ihrem Namen oder anderen Attributen.

pmap Gibt eine Speicherübersicht des angegebenen Prozesses aus.

ps Listet zur Zeit laufende Prozesse auf.

skill Sendet Signale an Prozesse, die den angegebenen Kriterien entsprechen.

slabtop Zeigt detaillierte Informationen zum Kernel-Slap-Cache in Echtzeit an.

snice Ändert die Priorität von Prozessen, die auf die angegebenen Kriterien passen.

sysctl Ändert Kernelparamter zur Laufzeit.

tload Gibt eine Grafik der aktuellen durchschnittlichen Systemlast aus.

top Zeigt eine Liste der Prozesse an, die am meisten CPU-Last erzeugen. Ermöglicht eine Übersicht über laufende Prozesse in Echtzeit.

uptime Gibt aus, wie lange ein System bereits läuft, wieviele Benutzer eingeloggt sind und wie hoch die Systemlast ist.

vmstat Erzeugt Statistiken zur Ausnutzung des virtuellen Speichers, gibt Informationen zu Prozessen,

	Speicher, Paging, Block-IO, traps und CPU-Aktivität aus.
w	Zeigt an, welche Benutzer gerade eingeloggt sind, wo, und seit wann.
watch	Führt ein Kommando immer wieder aus und gibt eine Bildschirmseite von seiner Ausgabe aus. So können Sie die Ausgabe eines Programms beobachten.
`libproc`	Enthält Funktionen, die von den meisten Programmen in diesem Paket benutzt werden.

6.20. Sed-4.1.5

Das Paket Sed enthält einen Stream-Editor.

Geschätzte Kompilierzeit: 0.1 SBU
Ungefähr benötigter Festplattenplatz: 6.4 MB

6.20.1. Installation von Sed

Bereiten Sie Sed zum Kompilieren vor:

```
./configure --prefix=/usr --bindir=/bin --enable-html
```

Die Bedeutung des neuen Parameters zu configure:

`--enable-html`
Dadurch wird die HTML-Dokumentation erzeugt.

Kompilieren Sie das Paket:

```
make
```

Um das Ergebnis zu prüfen, führen Sie **`make check`** aus.

Installieren Sie das Paket:

```
make install
```

6.20.2. Inhalt von Sed

Installiertes Programm: sed

Kurze Beschreibungen

sed Wird zum Filtern und Transformieren von Dateien in einem einzigen Durchlauf verwendet.

6.21. Libtool-1.5.22

Das Libtool-Skript enthält die Unterstützung für Bibliotheken. Libtool versteckt die Komplexität von gemeinsam benutzten Bibliotheken hinter einer konsistenten und portablen Schnittstelle.

Geschätzte Kompilierzeit: 0.1 SBU
Ungefähr benötigter Festplattenplatz: 16.6 MB

6.21.1. Installation von Libtool

Bereiten Sie Libtool zum Kompilieren vor:

```
./configure --prefix=/usr
```

Kompilieren Sie das Paket:

```
make
```

Um das Ergebnis zu prüfen, führen Sie **`make check`** aus.

Installieren Sie das Paket:

```
make install
```

6.21.2. Inhalt von Libtool

Installierte Programme: libtool und libtoolize
Installierte Bibliotheken: libltdl.{a,so}

Kurze Beschreibungen

libtool	Stellt vereinheitlichte Dienste zum Erstellen von Bibliotheken zur Verfügung.
libtoolize	Stellt einen einheitlichen Weg zur Verfügung um einem Paket **libtool**-Unterstützung hinzuzufügen.
`libltdl`	Versteckt die verschiedenen Schwierigkeiten mit Bibliotheken die dlopen verwenden.

6.22. Perl-5.8.8

Das Paket Perl enthält die Skriptsprache Perl (Practical Extraction and Report Language).

Geschätzte Kompilierzeit: 1.5 SBU
Ungefähr benötigter Festplattenplatz: 143 MB

6.22.1. Installation von Perl

Erstellen Sie nun eine grundlegende Version der Datei `/etc/hosts`. Diese wird in einer von Perls Konfigurationsdateien und in der Testsuite verwendet (falls Sie diese durchlaufen lassen).

```
echo "127.0.0.1 localhost $(hostname)" > /etc/hosts
```

Wenn Sie die vollständige Kontrolle darüber haben möchten, wie Perl sich selbst zum Installieren einrichtet, dann können Sie stattdessen das interaktive **Configure**-Skript benutzen. Wenn Sie mit den (normalerweise sinnvollen) von Perl automatisch erkannten Voreinstellungen zufrieden sind, benutzen Sie einfach das folgende Kommando:

```
./configure.gnu --prefix=/usr \
    -Dman1dir=/usr/share/man/man1 \
    -Dman3dir=/usr/share/man/man3 \
    -Dpager="/usr/bin/less -isR"
```

Die Bedeutung der configure-Parameter:

`-Dpager="/usr/bin/less -isR"`
: Dies korrigiert einen Fehler in der Art und Weise, wie **perldoc**, das Programm **less** aufruft.

`-Dman1dir=/usr/share/man/man1 -Dman3dir=/usr/share/man/man3`
: Da zur Zeit noch kein Groff installiert ist, geht **configure** davon aus, dass die Man-pages nicht erstellt werden sollen. Geben Sie diese Parameter ein, um die falsche Entscheidung zu übergehen.

Kompilieren Sie das Paket:

```
make
```

Zum Testen der Ergebnisse führen Sie dieses Kommando aus: **`make test`**.

Installieren Sie das Paket:

```
make install
```

6.22.2. Inhalt von Perl

Installierte Programme: a2p, c2ph, dprofpp, enc2xs, find2perl, h2ph, h2xs, instmodsh, libnetcfg, perl, perl5.8.8 (Link auf perl), perlbug, perlcc, perldoc, perlivp, piconv, pl2pm, pod2html, pod2latex, pod2man, pod2text, pod2usage, podchecker, podselect, psed (Link auf s2p), pstruct (Link auf c2ph), s2p, splain und xsubpp
Installierte Bibliotheken: Mehrere hundert, die hier nicht alle aufgelistet werden können

Kurze Beschreibungen

a2p	Übersetzt awk zu Perl.
c2ph	Gibt C-Strukturen aus, die von **cc -g -S** erzeugt wurden.
dprofpp	Zeigt Perl-Profiling-Daten an.
enc2xs	Erzeugt aus Unicode-Zeichenzuordnungen oder Tcl-Encoding-Dateien eine Perl-Erweiterung für das Encode-Modul.
find2perl	Übersetzt **find**-Kommandos zu Perl.
h2ph	Konvertiert `.h` C Header-Dateien zu `.ph` Perl Header-Dateien.
h2xs	Konvertiert `.h` C Header-Dateien zu Perl-Erweiterungen.
instmodsh	Ein Shell-Skript für den Umgang mit den installierten Perl-Module; es kann sogar ein Tar-Archiv aus einem installierten Modul erzeugen.
libnetcfg	Kann zum Einrichten von `libnet` benutzt werden.
perl	Kombiniert viele der besten Eigenschaften von C, **sed**, **awk** und **sh** in einer einzigen universell einsetzbaren Sprache. Perl wird auch als das Schweitzer Taschenmesser für Programmier bezeichnet.
perl5.8.8	Ein harter Link auf **perl**.
perlbug	Wird zum Erzeugen und Emailen von Fehlerberichten zu Perl oder seinen Modulen verwendet.
perlcc	Erzeugt ausführbare Dateien aus Perl-Programmen.
perldoc	Zeigt Teile einer Dokumentation im pod-Format an.
perlivp	Die Perl Installations-Prüfprozedur. Damit wird geprüft, ob Perl und seine Bibliotheken korrekt installiert wurden.
piconv	Die Perl-Version des Zeichensatz-Konverters **iconv**.
pl2pm	Ein Werkzeug zum groben Umwandeln von Perl4 `.pl`-Dateien in Perl5 `.pm`-Module.
pod2html	Konvertiert pod-Dateien in das HTML-Format.
pod2latex	Konvertiert pod-Dateien zu LaTeX.
pod2man	Konvertiert pod-Daten zu formatierter *roff-Eingabe.
pod2text	Konvertiert pod-Daten in formatierten ASCII-Text.
pod2usage	Gibt Benutzungshinweise aus eingebetteten pod-Dokumenten in Dateien aus.
podchecker	Prüft die Syntax von pod-Dokumentationsdateien.

podselect	Zeigt ausgewählte Abschnitte einer pod-Dokumentation an.
psed	Die Perl-Version des Stream-Editors **sed**.
pstruct	Gibt C-Strukturen aus, die von **cc -g -S** erzeugt wurden.
s2p	Konvertiert **sed**-Skripte zu perl.
splain	Erzwingt die ausführliche Analyse von Warnungen in Perl.
xsubpp	Konvertiert Perl XS-Code zu C-Code.

6.23. Readline-5.1

Das Paket Readline enthält Bibliotheken die Unterstützung für einen Verlauf und das Bearbeiten von Kommandozeilen bereitstellen.

Geschätzte Kompilierzeit: 0.1 SBU
Ungefähr benötigter Festplattenplatz: 10.2 MB

6.23.1. Installation von Readline

Die Upstream-Entickler haben seit der letzten Veröffentlichung von Readline-5.1 viele Fehler behoben. Wenden Sie diese Korrekturen an:

```
patch -Np1 -i ../readline-5.1-fixes-3.patch
```

Durch die Neuinstallation von Readline werden die alten Bibliotheken nach <bibliothek>.old umbenannt. Normalerweise ist das kein Problem, kann aber in einigen wenigen Fällen zu Linkerproblemen in **ldconfig** führen. Das Problem lässt sich mit den folgenden beiden seds umgehen:

```
sed -i '/MV.*old/d' Makefile.in
sed -i '/{OLDSUFF}/c:' support/shlib-install
```

Bereiten Sie Readline zum Kompilieren vor:

```
./configure --prefix=/usr --libdir=/lib
```

Kompilieren Sie das Paket:

```
make SHLIB_LIBS=-lncurses
```

Die Bedeutung der make-Option:

SHLIB_LIBS=-lncurses
: Dieser Parameter zwingt Readline, gegen die Bilbiothek `libncurses` zu linken (in Wirklichkeit natürlich `libncursesw`).

Dieses Paket enthält keine Testsuite.

Installieren Sie das Paket:

```
make install
```

Vergeben Sie Readline's dynamischen Bibliotheken passendere Zugriffsrechte:

```
chmod -v 755 /lib/lib{readline,history}.so*
```

Nun verschieben Sie die statischen Bibliotheken an eine passendere Stelle:

```
mv -v /lib/lib{readline,history}.a /usr/lib
```

Als nächstes werden die `.so`-Dateien im Ordner `/lib` gelöscht und nach `/usr/lib` verlinkt:

```
rm -v /lib/lib{readline,history}.so
ln -sfv ../../lib/libreadline.so.5 /usr/lib/libreadline.so
ln -sfv ../../lib/libhistory.so.5 /usr/lib/libhistory.so
```

6.23.2. Inhalt von Readline

Installierte Bibliotheken: libhistory.{a,so} und libreadline.{a,so}

Kurze Beschreibungen

`libhistory`	Stellt eine konsistente Schnittstelle zum Wiederaufrufen von Zeilen aus dem Verlauf zur Verfügung.
`libreadline`	Kümmert sich um die Konsistenz der Benutzerschnittstelle bei Programmen, die eine Kommandozeilenoberfläche bereitstellen müssen.

6.24. Zlib-1.2.3

Die in Zlib enthaltenen Routinen werden von vielen Programmen zum Komprimieren und Dekomprimieren genutzt.

Geschätzte Kompilierzeit: weniger als 0.1 SBU
Ungefähr benötigter Festplattenplatz: 3.1 MB

6.24.1. Installation von Zlib



Anmerkung

Vorsicht: Zlib baut seine gemeinsamen Bibliotheken falsch, wenn die Umgebungsvariable `CFLAGS` gesetzt ist. Falls Sie die Umgebungsvariable `CFLAGS` verwenden, fügen Sie ihr für den Durchlauf von **configure** den Wert *`-fPIC`* an und entfernen Sie ihn später wieder.

Bereiten Sie Zlib zum Kompilieren vor:

```
./configure --prefix=/usr --shared --libdir=/lib
```

Kompilieren Sie das Paket:

```
make
```

Um das Ergebnis zu prüfen, führen Sie **make check** aus.

Installieren Sie die gemeinsamen Bibliotheken:

```
make install
```

Das vorige Kommando hat eine `.so`-Datei im Ordner `/lib` installiert. Entfernen Sie sie wieder und erstellen Sie stattdessen einen Link in `/usr/lib`:

```
rm -v /lib/libz.so
ln -sfv ../../lib/libz.so.1.2.3 /usr/lib/libz.so
```

Kompilieren Sie nun die statische Bibliothek:

```
make clean
./configure --prefix=/usr
make
```

Um das Ergebnis zu prüfen, führen Sie **make check** aus.

Installieren Sie die statische Bibliothek:

```
make install
```

Und korrigieren Sie die Zugriffsrechte auf die statische Bibliothek:

```
chmod -v 644 /usr/lib/libz.a
```

6.24.2. Inhalt von Zlib

Installierte Bibliotheken: libz.{a,so}

Kurze Beschreibungen

`libz` Enthält Funktionen zum Komprimieren und Dekomprimieren, die von vielen Programmen genutzt werden.

6.25. Autoconf-2.59

Autoconf erstellt Shell-Skripte, mit denen man Software-Pakete automatisch zum Kompilieren einrichten kann.

Geschätzte Kompilierzeit: less than 0.1 SBU
Ungefähr benötigter Festplattenplatz: 7.2 MB

6.25.1. Installation von Autoconf

Bereiten Sie Autoconf zum Kompilieren vor:

```
./configure --prefix=/usr
```

Kompilieren Sie das Paket:

```
make
```

Um die Ergebnisse zu prüfen, geben Sie **make check** ein. Dies dauert allerdings ungefähr 3 SBUs. Außerdem werden 2 Tests übersprungen, die Automake verwenden. Wenn Sie den vollständigen Test durchführen lassen möchten, müssen Sie Autoconf nach der Installation von Automake erneut testen.

Installieren Sie das Paket:

```
make install
```

6.25.2. Inhalt von Autoconf

Installierte Programme: autoconf, autoheader, autom4te, autoreconf, autoscan, autoupdate und ifnames

Kurze Beschreibungen

autoconf Ein Werkzeug zum Erzeugen von Shell-Skripten, die Quellcode-Pakete automatisch einrichten und sie an unterschiedliche Unix-System anpassen. Die resultierenden configure-Skripte sind eigenständig—sie können auch dann ausgeführt werden, wenn **autoconf** nicht installiert ist.

autoheader Ein Werkzeug zum Erzeugen von Vorlagedateien für C *#define*-Anweisungen, die configure benutzen soll.

autom4te Ein Wrapper zu dem Makroprozessor M4.

autoreconf Führt automatisch **autoconf**, **autoheader**, **aclocal**, **automake**, **gettextize** und **libtoolize** in der richtigen Reihenfolge aus. Das spart Zeit, wenn Änderungen an **autoconf** und **automake** Vorlagedateien gemacht wurden.

autoscan Kann beim Erzeugen einer `configure.in`-Datei für ein Softwarepaket behilflich sein. Es untersucht die Quelldateien in einem Ordner und sucht nach üblichen Portabilitätsproblemen und erzeugt eine `configure.scan`-Datei, die als Basis für eine `configure.in`-Datei zu dem Softwarepaket dienen kann.

autoupdate Verändert eine `configure.in`-Datei so, dass sie nicht mehr die alten Namen der **autoconf** Makros aufruft, sondern die neuen.

ifnames Kann beim Schreiben einer `configure.in`-Datei für ein Paket hilfreich sein. Es gibt

die Bezeichner aus, die ein Paket in Präprozessor-Konditionen benutzt. Wenn ein Paket bereits für Portabilität eingerichtet ist, kann dieses kleine Werkzeug zum Auffinden der nötigen **configure**-Tests hilfreich sein. Es kann einige Lücken in `autoscan`-generierten `configure.in`-Dateien füllen.

6.26. Automake-1.9.6

Automake enthält Programme zur Erzeugung von Makefile-Dateien zur weiteren Verwendung mit Autoconf.

Geschätzte Kompilierzeit: less than 0.1 SBU
Ungefähr benötigter Festplattenplatz: 7.9 MB

6.26.1. Installation von Automake

Bereiten Sie Automake zum Kompilieren vor:

```
./configure --prefix=/usr
```

Kompilieren Sie das Paket:

```
make
```

Zum Testen der Ergebnisse können Sie **make check** benutzen. Dies dauert recht lange; etwa 10 SBUs.

Installieren Sie das Paket:

```
make install
```

6.26.2. Inhalt von Automake

Installierte Programme: acinstall, aclocal, aclocal-1.9.6, automake, automake-1.9.6, compile, config.guess, config.sub, depcomp, elisp-comp, install-sh, mdate-sh, missing, mkinstalldirs, py-compile, symlink-tree und ylwrap

Kurze Beschreibungen

acinstall	Ein Skript, das M4-Dateien im aclocal-Stil installiert.
aclocal	Erzeugt auf dem Inhalt von `configure.in`-Dateien basierend, entsprechende `aclocal.m4`-Dateien.
aclocal-1.9.6	Ein harter Link auf **aclocal**.
automake	Ein Werkzeug zum automatischen Erzeugen von `Makefile.in`'s aus sog. `Makefile.am`-Dateien. Um alle `Makefile.in`-Dateien eines Pakets zu erzeugen, lassen Sie dieses Programm im Basisordner des Pakets laufen. Durch das Scannen von `configure.in` findet es automatisch jede nötige `Makefile.am`-Datei und erzeugt die entsprechende `Makefile.in`-Datei.
automake-1.9.6	Ein harter Link auf **automake**.
compile	Ein Wrapper für verschiedene Compiler.
config.guess	Ein Skript. Es versucht, kanonische Tripplets für das Build, den Host oder die Zielarchitektur zu erraten.
config.sub	Ein Unter-Skript zum Validieren der Konfiguration.

depcomp	Ein Skript zum Kompilieren eines Programmes, so dass nicht nur das gewünschte Ergebnis erzeugt wird, sondern auch Abhängigkeitsinformationen generiert werden.
elisp-comp	Byte-kompiliert Emacs Lisp-Code.
install-sh	Ein Skript, welches ein Programm, ein Skript oder eine Datendatei installiert.
mdate-sh	Ein Skript, welches den Änderungszeitstempel einer Datei oder eines Ordners ausgibt.
missing	Ein Skript, welches fehlende GNU-Programme während der Installation ersetzt.
mkinstalldirs	Ein Skript zum Erzeugen einer Ordnerstruktur.
py-compile	Kompiliert ein Python-Programm.
symlink-tree	Ein Skript zum Erzeugen einer Symlink-Version einer Ordnerstruktur.
ylwrap	Ein Wrapper für **lex** und **yacc**.

6.27. Bash-3.1

Das Paket Bash enthält die Bourne-Again-SHell.

Geschätzte Kompilierzeit: 0.4 SBU
Ungefähr benötigter Festplattenplatz: 25.8 MB

6.27.1. Installation von Bash

Wenn Sie die Bash-Dokumentation heruntergeladen haben und die HTML-Dokumentation installieren möchten, dann führen Sie bitte die folgenden Kommandos aus:

```
tar -xvf ../bash-doc-3.1.tar.gz &&
sed -i "s|htmldir = @htmldir@|htmldir = /usr/share/doc/bash-3.1|" \
    Makefile.in
```

Die Upstream-Entwickler haben seit der ersten Veröffentlichung von Bash-3.1 viele Fehler behoben. Wenden Sie diese Fehlerkorrekturen nun an:

```
patch -Np1 -i ../bash-3.1-fixes-8.patch
```

Bereiten Sie Bash zum Kompilieren vor:

```
./configure --prefix=/usr --bindir=/bin \
    --without-bash-malloc --with-installed-readline
```

Die Bedeutung der configure-Parameter:

`--with-installed-readline`
: Dieser Parameter lässt Bash die von uns installierte `readline`-Bibliothek anstelle der Bash-eigenen Version benutzen.

Kompilieren Sie das Paket:

```
make
```

Zum Testen der Ergebnisse führen Sie dieses Kommando aus: **`make tests`**.

Installieren Sie das Paket:

```
make install
```

Starten Sie die frisch installierte **bash** (ersetzt die gerade laufende Version):

```
exec /bin/bash --login +h
```



Anmerkung

Die verwendeten Parameter machen **bash** zu einer interaktiven Login-Shell. Hashing bleibt weiterhin abgeschaltet, so dass frisch installierte Programme sofort verfügbar sind.

6.27.2. Inhalt von Bash

Installierte Programme: bash, bashbug und sh (Link auf bash)

Kurze Beschreibungen

bash Ein weit verbreiteter Befehlsinterpreter. Er führt alle möglichen Arten von Erweiterungen und Ersetzungen an einer Kommandozeile durch, bevor diese dann ausgeführt wird. Das macht diesen Befehlsinterpreter zu einem mächtigen Werkzeug.

bashbug Ein Shell-Skript, welches dem Benutzer helfen soll, einen Fehlerbericht zur **bash** in einem standardisierten Format zu erstellen und per E-Mail zu versenden.

sh Ein symbolischer Link auf das Programm **bash**. Wenn die **bash** als **sh** aufgerufen wird, versucht sie, das Verhalten der historischen Versionen von **sh** so gut wie möglich nachzuahmen und bleibt dabei trotzdem POSIX-Konform.

6.28. Bzip2-1.0.3

Das Paket Bzip2 enthält Programme zum Komprimieren und Dekomprimieren von Dateien. **Bzip2** erreicht vor allem bei Textdateien eine wesentlich bessere Kompressionsrate als das traditionelle **gzip**.

Geschätzte Kompilierzeit: weniger als 0.1 SBU
Ungefähr benötigter Festplattenplatz: 5.3 MB

6.28.1. Installation von Bzip2

Wenden Sie einen Patch an, um auch die Dokumentation zu diesem Paket zu installieren:

```
patch -Np1 -i ../bzip2-1.0.3-install_docs-1.patch
```

Das Programm **bzgrep** filtert die Buchstaben '|' und '&' in übergebenen Dateinamen nicht aus. Dadurch ist es möglich, Kommandos mit den Benutzerrechten des Benutzers von **bzgrep** auszuführen. Wenden Sie zur Korrektur diesen Patch an:

```
patch -Np1 -i ../bzip2-1.0.3-bzgrep_security-1.patch
```

bzdiff verwendet leider immer noch das missbilligte Programm **tempfile**. Wenden Sie diesen Patch an, damit anstelle dessn **mktemp** verwendet wird:

```
sed -i 's@tempfile -d /tmp -p bz@mktemp -p /tmp@' bzdiff
```

Bereiten Sie Bzip2 zum Kompilieren vor:

```
make -f Makefile-libbz2_so
make clean
```

Die Bedeutung des make-Parameters:

-f Makefile-libbz2_so
: Dieser Parameter veranlasst Bzip2 dazu, ein alternatives `Makefile` (in diesem Fall `Makefile-libbz2_so`) zu verwenden. Dieses erzeugt eine dynamische Bibliothek `libbz2.so` und verlinkt die Bzip2-Werkzeuge damit.

Kompilieren und testen Sie das Paket:

```
make
```

Falls Sie Bzip2 neu installieren müssen, müssen Sie zuerst **`rm -vf /usr/bin/bz*`** ausführen, ansonsten schlägt **make install** fehl.

Installieren Sie die Programme:

```
make install
```

Installieren Sie die ausführbare Datei **bzip2** nach `/bin`. Dann erzeuen Sie ein paar nötige symbolische Links und räumen auf:

```
cp -v bzip2-shared /bin/bzip2
cp -av libbz2.so* /lib
ln -sv ../../lib/libbz2.so.1.0 /usr/lib/libbz2.so
rm -v /usr/bin/{bunzip2,bzcat,bzip2}
ln -sv bzip2 /bin/bunzip2
ln -sv bzip2 /bin/bzcat
```

6.28.2. Inhalt von Bzip2

Installierte Programme: bunzip2 (Link auf bzip2), bzcat (Link auf bzip2), bzcmp, bzdiff, bzegrep, bzfgrep, bzgrep, bzip2, bzip2recover, bzless und bzmore
Installierte Bibliotheken: libbz2.{a,so}

Kurze Beschreibungen

bunzip2	Dekomprimiert bzip2-Dateien.
bzcat	Dekomprimiert zur Standardausgabe.
bzcmp	Führt **cmp** auf bzip2-Dateien aus.
bzdiff	Führt **diff** auf bzip2-Dateien aus.
bzgrep	Führt **grep** auf bzip2-Dateien aus.
bzegrep	Führt **egrep** auf bzip2-Dateien aus.
bzfgrep	Führt **fgrep** auf bzip2-Dateien aus.
bzip2	Komprimiert Dateien mit dem blocksortierenden Burrows-Wheeler Textkompressionsalgorythmus und Huffman-Kodierung. Die Kompressionsrate ist merkbar besser als die von herkömmlichen Kompressoren mit LZ77/LZ78, wie zum Beispiel **gzip**.
bzip2recover	Versucht, Daten aus beschädigten bzip2-Dateien zu reparieren.
bzless	Führt **less** auf bzip2-Dateien aus.
bzmore	Führt **more** auf bzip2-Dateien aus.
`libbz2*`	Die Bibliothek, die verlustlose blocksortierende Datenkompression mit Hilfe des Burrows-Wheeler-Algorythmus implementiert.

6.29. Diffutils-2.8.1

Die Programme dieses Pakets können Unterschiede zwischen Dateien oder Ordnern anzeigen.

Geschätzte Kompilierzeit: 0.1 SBU
Ungefähr benötigter Festplattenplatz: 6.3 MB

6.29.1. Installation von Diffutils

Nach POSIX muss **diff** mit weißen Leerzeichen Locale-spezifisch umgehen. Der folgende Patch behebt die Inkompatibilität zu dieser Regel:

```
patch -Np1 -i ../diffutils-2.8.1-i18n-1.patch
```

Der obige Patch hat den Nebeneffekt, dass die Man-page `diff.1` mit dem fehlenden Programm **help2man** neu erzeugt werden würde. Dies ergibt eine unleserliche Man-page für **diff**. Wir können das Problem vermeiden, indem wir den Zeitstempel von `man/diff.1` aktualisieren:

```
touch man/diff.1
```

Bereiten Sie Diffutils zum Kompilieren vor:

```
./configure --prefix=/usr
```

Kompilieren Sie das Paket:

```
make
```

Dieses Paket enthält keine Testsuite.

Installieren Sie das Paket:

```
make install
```

6.29.2. Inhalt von Diffutils

Installierte Programme: cmp, diff, diff3 und sdiff

Kurze Beschreibungen

cmp Vergleicht zwei Dateien und berichtet, ob, und an welchen Bytes sie sich unterscheiden.

diff Vergleicht zwei Dateien oder Ordner und berichtet, in welchen Zeilen sie sich unterscheiden.

diff3 Vergleicht drei Dateien Zeile für Zeile.

sdiff Führt interaktiv zwei Dateien zusammen und gibt das Ergebnis aus.

6.30. E2fsprogs-1.39

E2fsprogs stellt die Werkzeuge zur Verwendung mit dem `ext2`-Dateisystem zur Verfügung. Auch `ext3` wird unterstützt (ein Journaling Dateisystem).

Geschätzte Kompilierzeit: 0.4 SBU
Ungefähr benötigter Festplattenplatz: 31.2 MB

6.30.1. Installation von E2fsprogs

Es wird empfohlen, E2fsprogs außerhalb des Quellordners zu kompilieren:

```
mkdir -v build
cd build
```

Bereiten Sie E2fsprogs zum Kompilieren vor:

```
../configure --prefix=/usr --with-root-prefix="" \
    --enable-elf-shlibs --disable-evms
```

Die Bedeutung der configure-Parameter:

`--with-root-prefix=""`
Bestimmte Programme (wie z. B. **e2fsck**) sind absolut essentiell. Sie müssen z. B. selbst dann verfügbar sein, wenn `/usr` noch nicht eingehängt ist. Diese Programme gehören in Ordner wie `/lib` und `/sbin`. Ohne diese Option würden die Programme entgegen unserem Willen in `/usr` installiert werden.

`--enable-elf-shlibs`
Das erzeugt die gemeinsamen Bibliotheken, die einige Programme in diesem Paket verwenden.

`--disable-evms`
Dies deaktiviert die Installation des Enterprise Volume Management System (EVMS) Plugin. Das Plugin ist nicht auf dem Stand der aktuellen internen EVMS Schnittstellen und außerdem wird EVMS nicht als Teil des LFS Basis-Systems installiert; daher brauchen wir dieses Plugin nicht. Weitere Informationen erhalten Sie auf der Webseite von EMVS unter *http://evms.sourceforge.net/*.

Kompilieren Sie das Paket:

```
make
```

Um das Ergebnis zu prüfen, führen Sie **`make check`** aus.

Einer der Test von E2fsprogs wird 256MB Arbeitsspeicher beanspruchen. Wenn Sie nicht wesentlich mehr als 256MB Arbeitsspeicher habe, sollten Sie zumindest genügend Auslagerungsspeicher für diesen Test zur Verfügung haben. Lesen Sie unter Abschnitt 2.3, „Erstellen eines Dateisystems auf der neuen Partition“ und Abschnitt 2.4, „Einhängen (mounten) der neuen Partition“ nach, wie man Auslagerungsspeicher anlegt und aktiviert.

Installieren Sie die Binärdateien und die Dokumentation:

```
make install
```

Installieren Sie die gemeinsamen Bibliotheken:

```
make install-libs
```

6.30.2. Inhalt von E2fsprogs

Installierte Programme: badblocks, blkid, chattr, compile_et, debugfs, dumpe2fs, e2fsck, e2image, e2label, filefrag, findfs, fsck, fsck.ext2, fsck.ext3, logsave, lsattr, mk_cmds, mke2fs, mkfs.ext2, mkfs.ext3, mklost+found, resize2fs, tune2fs und uuidgen.
Installierte Bibliotheken: libblkid.{a,so}, libcom_err.{a,so}, libe2p.{a,so}, libext2fs.{a,so}, libss.{a,so} und libuuid.{a,so}

Kurze Beschreibungen

badblocks Durchsucht ein Gerät (üblicherweise eine Festplatte) nach defekten Blöcken.

blkid Ein Kommandozeilenprogramm zum Auffinden und Anzeigen der Eigenschaften eines Blockgerätes.

chattr Ändert Attribute eines `ext2`-Dateisystems. Auch `ext3` wird unterstützt (die Journaling-Version von `ext2`).

compile_et Ein Fehlertabellen-Compiler. Er konvertiert eine Tabelle mit Fehlercode-Namen und Meldungen zu einer C-Quelldatei, die dann mit der `com_err` Bibliothek verwendet werden kann.

debugfs Ein Dateisystemdebugger. Er kann benutzt werden, um den Status eines `ext2`-Dateisystems zu untersuchen und zu verändern.

dumpe2fs Gibt Informationen zum Superblock und zu Blockgruppen des Dateisystems auf einem bestimmten Gerät aus.

e2fsck Wird zum Prüfen und optional zum Reparieren von `ext2`- und `ext3`-Dateisystemen verwendet.

e2image Wird zum Speichern kritischer `ext2`-Dateisystemdaten in eine Datei verwendet.

e2label Zeigt oder verändert das Label eines `ext2`-Dateisystems auf dem angegebenen Gerät.

filefrag Berichtet über den Fragmentierungsstatus einer Datei

findfs Findet ein Dateisystem mit Hilfe des Label oder einer UUID (Universally Unique Identifier).

fsck Wird zum Prüfen und (optional) Reparieren eines Dateisystems verwendet.

fsck.ext2 Prüft in der Voreinstellung `ext2`-Dateisysteme.

fsck.ext3 Prüft in der Voreinstellung `ext3`-Dateisysteme.

logsave Speichert die Ausgabe eines Kommandos in eine Logdatei.

lsattr Listet Dateiattribute eines `ext2`-Dateisystems auf.

mk_cmds Konvertiert eine Tabelle mit Kommando-Namen und Hilfemeldungen zu C-Quellcode, der dann mit der `libss` Subsystem-Bibliothek verwendet werden kann.

mke2fs Erzeugt ein `ext2`- oder `ext3`-Dateisystem auf dem angegebenen Gerät.

mkfs.ext2 Erzeugt in der Voreinstellung ein `ext2`-Dateisystem.

mkfs.ext3 Erzeugt in der Voreinstellung ein `ext3`-Dateisystem.

mklost+found Wird benutzt, um den Ordner `lost+found` auf einem second extended Dateisystem

zu erzeugen. Es führt eine Vorzuweisung von Blöcken zu diesem Ordner durch, um damit **e2fsck** die Arbeit zu erleichtern.

resize2fs Kann zum Vergrößern oder Verkleinern eines `ext2`-Dateisystems verwendet werden.

tune2fs Wird zum Einstellen von veränderbaren Parametern auf einem `ext2`-Dateisystem eingesetzt.

uuidgen Erzeugt neue, universell einzigartige Bezeichner (UUID). Jede UUID kann grundsätzlich als einzigartig betrachtet werden, auf dem lokalen oder auf anderen Systemen, in der Vergangenheit und in der Zukunft.

`libblkid` Enthält Routinen zum Identifizieren von Geräten und zum Extrahieren von Token.

`libcom_err` Die allgemeine Routine zum Anzeigen von Fehlern.

`libe2p` Wird von **dumpe2fs**, **chattr** und **lsattr** benutzt.

`libext2fs` Enthält Routinen, die Programme im Benutzerkontext zum Manipulieren eines `ext2`-Dateisystems verwenden können.

`libss` Wird von **debugfs** benutzt.

`libuuid` Enthält Routinen zum Erzeugen von einmaligen Bezeichnern für Objekte, die hinter dem lokalen System verfügbar sein könnten.

6.31. File-4.17

File ist ein kleines Werkzeug mit dem man den Dateityp einer oder mehrerer Dateien feststellen kann.

Geschätzte Kompilierzeit: 0.1 SBU
Ungefähr benötigter Festplattenplatz: 7.5 MB

6.31.1. Installation von File

Bereiten Sie File zum Kompilieren vor:

```
./configure --prefix=/usr
```

Kompilieren Sie das Paket:

```
make
```

Dieses Paket enthält keine Testsuite.

Installieren Sie das Paket:

```
make install
```

6.31.2. Inhalt von File

Installierte Programme: file
Installierte Bibliothek: libmagic.{a,so}

Kurze Beschreibungen

file	Versucht, Dateien zu klassifizieren. Dazu führt es verschiedene Tests durch—Dateisystem-Tests, Tests mit „magischen“ Nummern, und Sprachtests. Der erste erfolgreiche Test entscheidet über das Ergebnis.
`libmagic`	Enthält Routinen zur Erkennung von „magischen“ Nummern; wird vom Programm **file** verwendet.

6.32. Findutils-4.2.27

Das Paket Findutils enthält Programme zum Auffinden von Dateien durch rekursive Suche in einer Ordnerstruktur oder über den Zugriff auf eine Datenbank. Die Suche über eine Datenbank ist normalerweise schneller, aber es besteht natürlich die Gefahr, dass die Datenbank zum Zeitpunkt der Suche veraltet ist.

Geschätzte Kompilierzeit: 0.2 SBU
Ungefähr benötigter Festplattenplatz: 12 MB

6.32.1. Installation von Findutils

Bereiten Sie Findutils zum Kompilieren vor:

```
./configure --prefix=/usr --libexecdir=/usr/lib/findutils \
    --localstatedir=/var/lib/locate
```

Die Bedeutung der configure-Parameter:

--localstatedir
Der obige Parameter ändert den Standort der **locate**-Datenbank wie vom FHS-Standard verlangt nach `/var/lib/locate`.

Kompilieren Sie das Paket:

```
make
```

Um das Ergebnis zu prüfen, führen Sie **make check** aus.

Installieren Sie das Paket:

```
make install
```

Einige der LFS-Bootskripte sing abhängig von dem Kommando **find**. Da `/usr` in den früheren Phasen des Bootvorgangs noch nicht einghängt sein könnte, muss sich dieses Programm auf der root-Partition befinden. Des Weiteren muss **updatedb** auf den neuen Pfad eingestellt werden:

```
mv -v /usr/bin/find /bin
sed -i -e 's/find:=${BINDIR}/find:=\/bin/' /usr/bin/updatedb
```

6.32.2. Inhalt von Findutils

Installierte Programme: bigram, code, find, frcode, locate, updatedb und xargs

Kurze Beschreibungen

bigram Wurde früher zum Anlegen von **locate**-Datenbanken benutzt.

code Wurde früher zum Anlegen von **locate**-Datenbanken benutzt. Es ist der Vorgänger von **frcode**.

find Durchsucht eine Ordnerstruktur nach Dateien, die einem bestimmten Kriterium entsprechen.

frcode Wird von **updatedb** aufgerufen, um die Liste der Dateinamen zu komprimieren. Durch die sog. front-Komprimierung wird die Datenbankgröße um den Faktor 4 bis 5 verkleinert.

locate Durchsucht die **locate**-Datenbank mit Dateinamen und gibt die Dateien aus, die eine bestimmte Zeichenkette enthalten oder auf ein bestimmtes Suchmuster passen.

updatedb Aktualisiert die **locate**-Datenbank. Es durchsucht das gesamte Dateisystem (inklusive anderer eingehängter Dateisysteme, wenn nicht anders angegeben) und trägt jeden gefundenen Dateinamen in die Datenbank ein.

xargs Kann benutzt werden, um ein bestimmtes Kommando auf eine Liste von Dateien anzuwenden.

6.33. Flex-2.5.33

Mit Flex kann man Programme zum Erkennen von Textmustern erzeugen.

Geschätzte Kompilierzeit: 0.1 SBU
Ungefähr benötigter Festplattenplatz: 8.4 MB

6.33.1. Installation von Flex

Bereiten Sie Flex zum Kompilieren vor:

```
./configure --prefix=/usr
```

Kompilieren Sie das Paket:

```
make
```

Um das Ergebnis zu prüfen, führen Sie **make check** aus.

Installieren Sie das Paket:

```
make install
```

Einige Programme erwarten die `lex`-Bibliothek in `/usr/lib`. Erstellen Sie daher einen entsprechenden symbolischen Link:

```
ln -sv libfl.a /usr/lib/libl.a
```

Einige wenige Programme kennen **flex** noch nicht und versuchen den Vorgänger **lex** aufzurufen. Um diesen Programmen dennoch gerecht zu werden, erzeugen Sie ein kleines Shell-Skript mit dem Namen `lex`, welches `flex` im **lex**-Emulationsmodus aufruft:

```
cat > /usr/bin/lex << "EOF"
#!/bin/sh
# Begin /usr/bin/lex

exec /usr/bin/flex -l "$@"

# End /usr/bin/lex
EOF
chmod -v 755 /usr/bin/lex
```

6.33.2. Inhalt von Flex

Installierte Programme: flex und lex
Installierte Bibliothek: libfl.a

Kurze Beschreibungen

flex Ein Werkzeug zum Erzeugen von Programmen, die Muster in Text erkennen können. Mustererkennung ist in vielen Programmen nützlich. Flex erzeugt aus einem Satz an Suchregeln ein Programm, das nach diesen Mustern sucht.

lex Ein Skript, welches **flex** im **lex**-Emulationsmodus startet.

`libfl.a` Die `flex`-Bibliothek.

6.34. GRUB-0.97

Das Paket Grub enthält den GRand Unified Bootloader.

Geschätzte Kompilierzeit: 0.2 SBU
Ungefähr benötigter Festplattenplatz: 10.2 MB

6.34.1. Installation von GRUB

Dieses Paket funktioniert unter Umständen nicht fehlerfrei, wenn die voreingestellten Optionen für Compiler-Optimierungen übergangen werden. (Dazu gehören auch *-march* und *-mcpu*.) Daher sollten die entsprechenden Umgebungsvariablen (wie z. B. `CFLAGS` und `CXXFLAGS`) für den Kompiliervorgang zurückgesetzt oder entsprechend abgeändert werden.

Beginnen Sie mit dem folgenden Patch zur besseren Erkennung von Laufwerken, Behebung einiger Probleme mit GCC 4.x und zur besseren SATA-Unterstützung für einige Festplattencontroller:

```
patch -Np1 -i ../grub-0.97-disk_geometry-1.patch
```

Bereiten Sie GRUB zum Kompilieren vor:

```
./configure --prefix=/usr
```

Kompilieren Sie das Paket:

```
make
```

Um das Ergebnis zu prüfen, führen Sie **`make check`** aus.

Installieren Sie das Paket:

```
make install
mkdir -v /boot/grub
cp -v /usr/lib/grub/i386-pc/stage{1,2} /boot/grub
```

Ersetzen Sie `i386-pc` durch den für Ihre Plattform korrekten Ordner.

Der Ordner `i386-pc` enthält auch einige `*stage1_5`-Dateien, die jeweils für verschiedene Dateisysteme gedacht sind. Schauen Sie nach, welche zur Verfügung stehen und kopieren Sie die notwendigen nach `/boot/grub`. Die meisten Leute werden `e2fs_stage1_5` und/oder `reiserfs_stage1_5` kopieren.

6.34.2. Inhalt von GRUB

Installierte Programme: grub, grub-install, grub-md5-crypt, grub-set-default, grub-terminfo und mbchk

Kurze Beschreibungen

grub	Die GRand Unified Bootloader Kommando-Shell.
grub-install	Installiert GRUB auf dem angegebenen Gerät.
grub-md5-crypt	Verschlüsselt Passwörter im MD5-Format.
grub-set-default	Stellt den Voreingestellten Boot-Eintrag für GRUB ein.
grub-terminfo	Erzeugt ein terminfo-Kommando aus dem Namen eines Terminals. Es kann

verwendet werden, wenn Sie ein unbekanntes Terminal haben.

mbchk Prüft das Format eines Multiboot-Kernel.

6.35. Gawk-3.1.5

Gawk ist eine Implementierung von awk und wird zur Textmanipulation verwendet.

Geschätzte Kompilierzeit: 0.2 SBU
Ungefähr benötigter Festplattenplatz: 18.2 MB

6.35.1. Installation von Gawk

Unter bestimmten Umständen gibt Gawk-3.1.5 einen Speicherblock frei, der gar nicht zugewiesen war. Mit dem folgenden Patch wird das Problem behoben:

```
patch -Np1 -i ../gawk-3.1.5-segfault_fix-1.patch
```

Bereiten Sie Gawk zum Kompilieren vor:

```
./configure --prefix=/usr --libexecdir=/usr/lib
```

Aufgrund eines Fehlers im **configure**-Skript erkennt Gawk einige Funktionen von Glibc's locale-Unterstützung nicht richtig. Das führt z. B. zu Fehlern in der Testsuite von Gettext. Sie können das Problem umgehen, indem Sie die fehlenden Makro-Definitionen in der Datei `config.h` hinzufügen:

```
cat >>config.h <<"EOF"
#define HAVE_LANGINFO_CODESET 1
#define HAVE_LC_MESSAGES 1
EOF
```

Kompilieren Sie das Paket:

```
make
```

Um das Ergebnis zu prüfen, führen Sie **`make check`** aus.

Installieren Sie das Paket:

```
make install
```

6.35.2. Inhalt von Gawk

Installierte Programme: awk (Link auf gawk), gawk, gawk-3.1.5, grcat, igawk, pgawk, pgawk-3.1.5 und pwcat

Kurze Beschreibungen

awk	Ein Link auf **gawk**.
gawk	Ein Programm zur Manipulation von Textdateien. Es ist die GNU-Implementierung von **awk**.
gawk-3.1.5	Ein harter Link auf **gawk**.
grcat	Zeigt die Gruppendatenbank `/etc/group` an.
igawk	Ermöglicht **gawk** das Einbinden von Dateien.
pgawk	Die Profiling-Version von **gawk**.

pgawk-3.1.5 Ein harter Link auf **pgawk**.

pwcat Zeigt die Passwortdatenbank `/etc/passwd` an.

6.36. Gettext-0.14.5

Gettext wird zur Übersetzung und Lokalisierung verwendet. Programme können mit Unterstützung für NLS (Native Language Support, Unterstützung für die lokale Sprache) kompiliert werden. Dadurch können Texte und Meldungen in der Sprache des Anwenders ausgegeben werden.

Geschätzte Kompilierzeit: 1 SBU
Ungefähr benötigter Festplattenplatz: 65 MB

6.36.1. Installation von Gettext

Bereiten Sie Gettext zum Kompilieren vor:

```
./configure --prefix=/usr
```

Kompilieren Sie das Paket:

```
make
```

Zum Durchlaufen der Testsuite können Sie dieses Kommando benutzen: **make check**. Leider benötigt diese Testsuite viel Zeit: etwa 5 SBU.

Installieren Sie das Paket:

```
make install
```

6.36.2. Inhalt von Gettext

Installierte Programme: autopoint, config.charset, config.rpath, envsubst, gettext, gettext.sh, gettextize, hostname, msgattrib, msgcat, msgcmp, msgcomm, msgconv, msgen, msgexec, msgfilter, msgfmt, msggrep, msginit, msgmerge, msgunfmt, msguniq, ngettext und xgettext
Installierte Bibliotheken: libasprintf.{a,so}, libgettextlib.so, libgettextpo.{a,so} und libgettextsrc.so

Kurze Beschreibungen

autopoint	Kopiert die Dateien einer typischen Gettext-Infrastruktur in ein Quellpaket.
config.charset	Gibt eine systemabhängige Tabelle von zeichenkodierenden Aliasen aus.
config.rpath	Gibt einen systemabhängigen Satz von Variablen aus, die beschreiben, wie der Laufzeit-Suchpfad von gemeinsamen Bibliotheken in einer ausführbaren Datei gesetzt wird.
envsubst	Erweitert Umgebungsvariablen in Shell-Format-Zeichenketten.
gettext	Übersetzt Nachrichten in natürlicher Sprache in die Muttersprache des Anwenders. Dafür benutzt es einen Übersetzungsnachrichten-Katalog.
gettext.sh	Dies ist hauptsächlich eine Bibltiothek mit Shell-Funktionen für Gettext.
gettextize	Kopiert alle standard-Gettext-Dateien in den Basisordner eines Pakets um so die ersten Schritte der Internationalisierung zu erleichtern.
hostname	Zeigt den Netzwerk-Hostnamen in verschiedenen Formen an.
msgattrib	Filtert Nachrichten in einem Übersetzungskatalog nach ihren Attributen und

manipuliert diese Attribute.

msgcat Fügt die angegebenen `.po`-Dateien aneinander und verschmelzt sie.

msgcmp Vergleicht zwei `.po`-Dateien, um sicherzustellen, dass beide den gleichen Satz an msgid-Zeichenketten enthalten.

msgcomm Findet die Nachrichten, die die angegebenen `.po`-Dateien gemeinsam haben.

msgconv Konvertiert den Übersetzungskatalog in einen anderen Zeichensatz.

msgen Erzeugt einen englischen Übersetzungskatalog.

msgexec Führt ein Kommando auf allen Übersetzungen in einem Katalog aus.

msgfilter Wendet einen Filter auf alle Übersetzungen in einem Katalog an.

msgfmt Erzeugt aus einem Übersetzungskatalog einen binären Katalog.

msggrep Extrahiert alle Nachrichten aus einem Katalog, die auf ein bestimmtes Muster passen oder zu einer bestimmten Quelldatei gehören.

msginit Erzeugt eine neue `.po`-Datei und initialisiert die Meta-Informationen mit Werten aus der Arbeitsumgebung des Benutzers.

msgmerge Kombiniert zwei Übersetzungen in eine einzige Datei.

msgunfmt Erzeugt aus einem binären Katalog einen Nachrichtenkatalog in Textform.

msguniq Vereinheitlicht doppelte Übersetzungen in einem Nachrichtenkatalog.

ngettext Zeigt die Übersetzung einer Textnachricht an, deren Grammatik von einer Zahl abhängt.

xgettext Extrahiert alle übersetzbaren Nachrichten aus den angegebenen Quelldateien, um daraus eine erste Nachrichtenkatalogvorlage zu erstellen.

`libasprintf` Definiert die *autosprintf*-Klasse; sie macht C-formatierte Routinen in C++ Programmen verfügbar, vor allem zur Verwendung mit <*string*> Strings und den <*iostream*> Streams.

`libgettextlib` Eine private Bibliothek, die die allgemeinen Routinen der verschiedenen gettext-Programme enthält. Sie sind nicht zur normalen Verwendung gedacht.

`libgettextpo` Wird zum Schreiben von spezialisierten Programmen verwendet, die `.po`-Dateien verarbeiten sollen. Diese Bibliothek wird benutzt, wenn die mitgelieferten Standardprogramme von **gettext** nicht ausreichen (so wie **msgattrib** und **msgen**).

`libgettextsrc` Eine private Bibliothek, die die allgemeinen Routinen der verschiedenen gettext-Programme enthält. Sie sind nicht zur normalen Verwendung gedacht.

6.37. Grep-2.5.1a

Das Paket Grep enthält Programme zum Durchsuchen von Dateien.

Geschätzte Kompilierzeit: 0.1 SBU
Ungefähr benötigter Festplattenplatz: 4.8 MB

6.37.1. Installation von Grep

Die aktuelle Version von Grep hat leider viele Fehler, insbesondere in der Unterstützung von Multibyte-Locales. RedHat behebt mit dem folgenden Patch zumindest einige dieser Fehler:

```
patch -Np1 -i ../grep-2.5.1a-redhat_fixes-2.patch
```

Damit die vom Patch hinzugefügten Tests erfolgreich durchlaufen können, müssen die Zugriffsrechte für die Testdatei geändert werden:

```
chmod +x tests/fmbtest.sh
```

Bereiten Sie Grep zum Kompilieren vor:

```
./configure --prefix=/usr --bindir=/bin
```

Kompilieren Sie das Paket:

```
make
```

Um das Ergebnis zu prüfen, führen Sie **`make check`** aus.

Installieren Sie das Paket:

```
make install
```

6.37.2. Inhalt von Grep

Installierte Programme: egrep (Link auf grep), fgrep (Link auf grep) und grep

Kurze Beschreibungen

egrep Gibt die Zeilen aus, die auf einen bestimmten regulären Ausdruck passen.

fgrep Gibt die Zeilen aus, die auf eine Liste von festgelegten Zeichenketten passen.

grep Gibt die Zeilen aus, die auf einen bestimmten einfachen regulären Ausdruck passen.

6.38. Groff-1.18.1.1

Groff enthält verschiedene Programme zur Verarbeitung und Formatierung von Text.

Geschätzte Kompilierzeit: 0.4 SBU
Ungefähr benötigter Festplattenplatz: 39.2 MB

6.38.1. Installation von Groff

Dieser Patch fügt Unterstützung für „ascii8“- und „nippon“-Geräte zu Groff hinzu:

```
patch -Np1 -i ../groff-1.18.1.1-debian_fixes-1.patch
```



Anmerkung

Diese Geräte werden von Man-DB beim Formatieren von nicht-englischen Man-pages verwendet, die nicht in der Kodierung ISO-8859-1 vorliegen. Derzeit gibt es keinen funktionierenden Patch für Groff-1.19.x, der diese Funktionalität hinzufügt.

Einige Bildschirmschriften enthalten nicht die Unicode-Variante der einfachen Anführungszeichen und Bindestriche. Stattdessen soll Groff die ASCII-Versionen verwenden:

```
sed -i -e 's/2010/002D/' -e 's/2212/002D/' \
    -e 's/2018/0060/' -e 's/2019/0027/' font/devutf8/R.proto
```

Groff erwartet, dass die Umgebungsvariable `PAGE` die Standardpapiergröße enthält. Für Anwender in den Vereinigten Staaten ist *`PAGE=letter`* korrekt. Wenn Ihr Standort woanders ist, ersetzen Sie bitte *`PAGE=letter`* durch *`PAGE=A4`*. Die Voreinstellung der Papiergröße wird zwar zum Kompilierzeitpunkt eingestellt werden. Jedoch kann man auch später noch in der Datei `/etc/papersize` die Papiergröße einstellen. Dazu müssen Sie nur „A4“ oder „letter“ in die Datei schreiben.

Bereiten Sie Groff zum Kompilieren vor:

```
PAGE=<papier_größe> ./configure --prefix=/usr --enable-multibyte
```

Kompilieren Sie das Paket:

```
make
```

Dieses Paket enthält keine Testsuite.

Installieren Sie das Paket:

```
make install
```

Einige Dokumentationsprogramme wie zum Beispiel **xman** funktionieren ohne diese symbolischen Links nicht:

```
ln -sv eqn /usr/bin/geqn
ln -sv tbl /usr/bin/gtbl
```

6.38.2. Inhalt von Groff

Installierte Programme: addftinfo, afmtodit, eqn, eqn2graph, geqn (Link auf eqn), grn, grodvi, groff, groffer, grog, grolbp, grolj4, grops, grotty, gtbl (Link auf tbl), hpftodit, indxbib, lkbib, lookbib, mmroff, neqn, nroff, pfbtops, pic, pic2graph, post-grohtml, pre-grohtml, refer, soelim, tbl, tfmtodit und troff

Kurze Beschreibungen

addftinfo Liest eine troff-Schriftdatei und fügt einige schriftmetrische Informationen hinzu, die vom **groff**-System benutzt werden.

afmtodit Erzeugt eine Schrift-Datei zur Verwendung mit **groff** und **grops**.

eqn Kompiliert in troff-Eingabedateien enthaltene Beschreibungen von Gleichungen zu Kommandos, die **troff** versteht.

eqn2graph Konvertiert eine EQN-Gleichung zu einem beschnittenen Bild.

geqn Ein Link auf **gawk**.

grn Ein **groff**-Präprozessor für gremlin-Dateien.

grodvi Ein Treiber für **groff**, der das TeX dvi-Format erzeugt.

groff Eine Benutzerschnittstelle für das groff-Dokumentenformatierungssystem. Normalerweise führt es das Programm **troff** und einen für das Ausgabegerät passenden Postprozessor aus.

groffer Zeigt groff-Dateien und Man-pages unter X und im tty an.

grog Liest Dateien ein und schätzt, welche der **groff**-Optionen `-e`, `-man`, `-me`, `-mm`, `-ms`, `-p`, `-s` und `-t` zum Drucken benötigt werden, und gibt das nötige **groff**-Kommando aus.

grolbp Ein **groff**-Treiber für Canon CAPSL-Drucker (Laserdrucker der Serie LBP-4 und LBP-8).

grolj4 Ein Treiber für **groff**, der Ausgaben im PCL5-Format, passend für HP LaserJet 4-Drucker erzeugt.

grops Übersetzt die Ausgabe von GNU **troff** zu PostScript.

grotty Übersetzt die Ausgabe von GNU **troff** in eine passende Form für schreibmaschinenähnliche Geräte.

gtbl Ein Link auf **tbl**.

hpftodit Erzeugt aus einer HP-markierten Schriftmetrik-Datei eine Schriftdatei zur Verwendung mit **groff -Tlj4**.

indxbib Erzeugt mit einer angegebenen Datei einen invertierten Index für die bibliographischen Datenbanken zur Verwendung mit **refer**, **lookbib** und **lkbib**.

lkbib Durchsucht bibliographische Datenbanken nach Referenzen, die bestimmte Schlüssel enthalten, und gibt die gefundenen Referenzen aus.

lookbib Gibt einen Prompt auf die standard-Fehlerausgabe (solange die Standardeingabe kein Terminal ist), liest eine Zeile mit Stichwörtern von der Standardeingabe, durchsucht eine bibliographische Datenbank nach Referenzen zu diesen Stichwörtern, gibt die gefundenen Referenzen aus und wiederholt das so lange bis keine weitere Eingabe mehr

vorhanden ist.

mmroff Ein einfacher Präprozessor für **groff**.

neqn Formatiert Gleichungen für die ASCII-Ausgabe (American Standard Code for Information Interchange).

nroff Ein Skript, das **nroff**-Kommandos mit **groff** emuliert.

pfbtops Übersetzt eine Postscript-Schrift im `.pfb`-Format zu ASCII.

pic Kompiliert in groff- oder TeX-Eingabedateien enthaltene Beschreibungen von Bildern zu Kommandos, die von TeX oder **troff** verwendet werden können.

pic2graph Konvertiert ein PIC-Diagramm zu einem beschnittenen Bild.

post-grohtml Übersetzt die Ausgabe von GNU **troff** zu HTML.

pre-grohtml Übersetzt die Ausgabe von GNU **troff** zu HTML.

refer Kopiert den Inhalt einer Datei zur Standardausgabe, außer das Zeilen zwischen *.[* und *.]* als Zitat interpretiert werden und Zeilen zwischen *.R1* und *.R2* als Kommandos behandelt werden, die angeben, wie mit Zitaten umgegangen werden soll.

soelim Liest Dateien und ersetzt Zeilen der Form *.so <Datei>* > mit dem tatsächlichen Inhalt von *<Datei>*.

tbl Kompiliert in troff-Eingabedateien eingebettete Beschreibungen von Tabellen zu Kommandos, die von **troff** unterstützt werden.

tfmtodit Erzeugt Schriftdateien zur Verwendung mit **groff -Tdvi**.

troff Ist hochkompatibel mit Unix **troff**. Üblicherweise wird es mit dem Kommando **groff** aufgerufen, welches auch Präprozessoren und Postprozessoren in der richtigen Reihenfolge und mit den richtigen Optionen aufruft.

6.39. Gzip-1.3.5

Das Paket Gzip enthält Programme zum Komprimieren und Dekomprimieren von Dateien.

Geschätzte Kompilierzeit: weniger als 0.1 SBU
Ungefähr benötigter Festplattenplatz: 2.2 MB

6.39.1. Installation von Gzip

Gzip hat zwei bekannte Sicherheitsmängel. Der folgende Patch behebt beide Probleme:

```
patch -Np1 -i ../gzip-1.3.5-security_fixes-1.patch
```

Bereiten Sie Gzip zum Kompilieren vor:

```
./configure --prefix=/usr
```

Das Skript **gzexe** hat den Pfad zu **gzip** fest eingebaut. Da diese Datei im nachhinein verschoben wird, müssen Sie mit dem folgenden Kommando sicherstellen, dass der korrekte Pfad in das Skript geschrieben wird:

```
sed -i 's@"BINDIR"@/bin@g' gzexe.in
```

Kompilieren Sie das Paket:

```
make
```

Dieses Paket enthält keine Testsuite.

Installieren Sie das Paket:

```
make install
```

Verschieben Sie die Datei **gzip** nach `/bin` und erzeugen Sie ein paar übliche Links dorthin:

```
mv -v /usr/bin/gzip /bin
rm -v /usr/bin/{gunzip,zcat}
ln -sv gzip /bin/gunzip
ln -sv gzip /bin/zcat
ln -sv gzip /bin/compress
ln -sv gunzip /bin/uncompress
```

6.39.2. Inhalt von Gzip

Installierte Programme: compress (Link auf gzip), gunzip (Link auf gzip), gzexe, gzip, uncompress (Link auf gunzip), zcat (Link auf gzip), zcmp, zdiff, zegrep, zfgrep, zforce, zgrep, zless, zmore und znew

Kurze Beschreibungen

compress	Komprimiert und Dekomprimiert Dateien.
gunzip	Dekomprimiert gzip-Dateien.
gzexe	Erzeugt selbstextrahierende ausführbare Dateien.
gzip	Komprimiert Dateien mit dem Lempel-Ziv (LZ77) Algorithmus.

uncompress Entpackt komprimierte Dateien.

zcat Dekomprimiert gzip-Dateien zur Standardausgabe.

zcmp Führt **cmp** auf gzip-Dateien aus.

zdiff Führt **diff** auf gzip-Dateien aus.

zegrep Führt **egrep** auf gzip-Dateien aus.

zfgrep Führt **fgrep** auf gzip-Dateien aus.

zforce Erzwingt eine `.gz`-Erweiterung an die komprimierten Dateien, damit **gzip** diese Dateien nicht erneut komprimiert. Das kann sinnvoll sein, wenn Dateinamen bei einer Datenübertragung abgeschnitten wurden.

zgrep Führt **grep** auf gzip-Dateien aus.

zless Führt **less** auf gzip-Dateien aus.

zmore Führt **more** auf gzip-Dateien aus.

znew Konvertiert Dateien im **compress**-Format in das **gzip**-Format—`.Z` zu `.gz`.

6.40. Inetutils-1.4.2

Inetutils enthält verschiedene Programme zur grundlegenden Netzwerkunterstützung.

Geschätzte Kompilierzeit: 0.2 SBU
Ungefähr benötigter Festplattenplatz: 8.9 MB

6.40.1. Installation von Inetutils

Wenden Sie einen Patch für Inetutils zum Kompilieren mit GCC-4.0.3 an:

```
patch -Np1 -i ../inetutils-1.4.2-gcc4_fixes-3.patch
```

Sie werden nicht alle Programme aus diesem Paket installieren. Dennoch würde Inetutils die Man-pages zu diesen Programmen installieren. Der folgende Patch behebt das Problem:

```
patch -Np1 -i ../inetutils-1.4.2-no_server_man_pages-1.patch
```

Bereiten Sie Inetutils zum Kompilieren vor:

```
./configure --prefix=/usr --libexecdir=/usr/sbin \
    --sysconfdir=/etc --localstatedir=/var \
    --disable-logger --disable-syslogd \
    --disable-whois --disable-servers
```

Die Bedeutung der configure-Parameter:

--disable-logger
: Das verhindert die Installation des Programmes **logger**, welches Nachrichten an den System-Log-Daemon übergibt. Logger wird hier ausgelassen, weil etwas später durch Util-Linux eine bessere Version installiert wird.

--disable-syslogd
: Dieser Parameter verhindert die Installation des System-Log-Daemon, weil Sie später einen anderen mit dem Paket Sysklogd installieren werden.

--disable-whois
: Dies verhindert die Installation des **whois**-Clients, welcher leider elendig veraltet ist. Im BLFS-Buch finden Sie eine Installations-Anleitung für einen besseren **whois**-Client.

--disable-servers
: Das verhindert die Installation verschiedener Server-Dienste die zu Inetutils gehören. Diese Dienste sind in einem Basis-System wie LFS nicht angebracht. Einige sind von Natur aus unsicher und nur in vertrauenswürdigen Netzen ohne Risiko einsetzbar. Mehr Informationen finden Sie unter *http://www.linuxfromscratch.org/blfs/view/svn/basicnet/inetutils.html*. Bitte beachten Sie auch, dass es für fast alle dieser Dienste einen besseren Ersatz gibt.

Kompilieren Sie das Paket:

```
make
```

Dieses Paket enthält keine Testsuite.

Installieren Sie das Paket:

```
make install
```

Und verschieben Sie das Programm **ping** an die richtige Stelle:

```
mv -v /usr/bin/ping /bin
```

6.40.2. Inhalt von Inetutils

Installierte Programme: ftp, ping, rcp, rlogin, rsh, talk, telnet und tftp

Kurze Beschreibungen

ftp	Das Programm für FTP (File Transfer Protocol).
ping	Sendet echo-request-Pakete und berichtet, wie lange die Antwort braucht.
rcp	Kopiert Dateien auf entfernten Systemen.
rlogin	Führt eine entfernte Anmeldung durch.
rsh	Führt eine entfernte Shell aus.
talk	Wird zum Unterhalten mit anderen Benutzern verwendet.
telnet	Dies ist ein Telnet-Client.
tftp	Das Programm zu TFTP (Trivial File Tranfer Protocol).

6.41. IPRoute2-2.6.16-060323

Das Paket IPRoute2 enthält verschiedene Programme zur grundlegenden Unterstützung von IPv4-basierten Netzwerken.

Geschätzte Kompilierzeit: 0.2 SBU
Ungefähr benötigter Festplattenplatz: 4.8 MB

6.41.1. Installation von IPRoute2

Kompilieren Sie das Paket:

```
make SBINDIR=/sbin
```

Die Bedeutung der make-Option:

SBINDIR=/sbin
: Dies stellt sicher, dass die Binärdateien von IPRoute2 nach /sbin installiert werden. Lt. FHS ist dies der korrekte Ort, weil einige der Programme aus IPRoute2 in Bootskripten verwendung finden.

Dieses Paket enthält keine Testsuite.

Installieren Sie das Paket:

```
make SBINDIR=/sbin install
```

Das Programm **arpd** verlinkt gegen die Berkely DB-Bibliotheken, die in /usr liegen und verwendet eine Datenbank in /var/lib/arpd/arpd.db. Nach FHS muss es aber in /usr/sbin liegen, also verschieben Sie es:

```
mv -v /sbin/arpd /usr/sbin
```

6.41.2. Inhalt von IPRoute2

Installierte Programme: arpd, ctstat (Link auf lnstat), ifcfg, ifstat, ip, lnstat, nstat, routef, routel, rtacct, rtmon, rtpr, rtstat (Link auf lnstat), ss und tc.

Kurze Beschreibungen

arpd Ein Userspace-ARP-Daemon. Er ist in sehr großen Netzwerken nützlich, wenn der Kernel-ARP-Daemon nicht ausreicht oder wenn man einen Honeypot für Sicherheitszwecke einrichten möchte.

ctstat Ein Werkzeug für den Verbindungsstatus.

ifcfg Ein Shell-Skript Wrapper für **ip**.

ifstat Zeigt Schnittstellenstatistiken an, inklusive der Menge der gesendeten und empfangenen Pakete pro Schnittstelle.

ip Dies ist die eigentliche ausführbare Datei. Sie hat viele Funktionen:

ip link *<Gerät>* zeigt den Gerätestatus an und ermöglicht Änderungen an den Einstellungen.

ip addr zeigt Adressen und ihre Eigenschaften an, fügt neue Adressen hinzu und löscht alte.

ip neighbor zeigt Bindungen und Eigenschaften von benachbarten Geräten an, fügt neue Nachbargerätebindungen hinzu und löscht alte.

ip rule zeigt Routingregeln an und bearbeitet sie.

ip route ermöglicht das Anzeigen und Ändern von Routingtabellen.

ip tunnel zeigt IP-Tunnel und die Eigenschaften an und ermöglicht Änderungen daran.

ip maddr zeigt Multicast-Adressen und ihre Eigenschaften an und ermöglicht Änderungen.

ip mroute setzt, ändert oder löscht Multicast-Routen.

ip monitor ermöglicht, dauerhaft den Status von Netzwerkgeräten, Adressen und Routen zu überwachen.

lnstat Bietet Netzwerkstatistiken unter Linux. Dies ist ein allgemeinerer und vollständigerer Ersatz für das alte Programm **rtstat**.

nstat Zeigt Netzwerkstatistiken an.

routef Eine Komponente von **ip route**. Sie wird zum Leeren der Routingtabellen genutzt.

routel Eine Komponente von **ip route**. Sie wird zum Auflisten der Routingtabellen genutzt.

rtacct Zeigt den Inhalt von `/proc/net/rt_acct` an.

rtmon Ein Werkzeug zum Überwachen des Routing.

rtpr Konvertiert die Ausgabe von **ip -o** zurück in eine lesbare Form.

rtstat Ein Werkzeug für den Routingstatus.

ss Ähnlich wie das Kommando **netstat**. Zeigt aktive Verbindungen an.

tc Programm zur Kontrolle des Netzwerkverkehrs (Traffic Controlling). Implementiert Quality of Service (QOS) und Class Of Service (COS):

tc qdisc ermöglicht das Einstellen der Warteschlangen-Regeln.

tc class ermöglicht das Einrichten von Klassen, basierend auf einer Warteschlangen-Regelung.

tc estimator ermöglicht das Schätzen des Netzwerk-Flusses in ein Netzwerk.

tc filter ermöglicht das Erstellen von QOS/COS Paketfiltern.

tc policy ermöglicht das Erstellen von QOS/COS Regelwerken.

6.42. Kbd-1.12

Kbd enthält die Dateien für das Tastaturlayout und entsprechende Werkzeuge dazu.

Geschätzte Kompilierzeit: weniger als 0.1 SBU
Ungefähr benötigter Festplattenplatz: 12.3 MB

6.42.1. Installation von Kbd

Das Verhalten der Tasten Backspace und Entfernen ist in den Tastaturlayouttabellen von Kbd nicht einheitlich geregelt. Der folgende Patch behebt das Problem für die i386-Tabellen:

```
patch -Np1 -i ../kbd-1.12-backspace-1.patch
```

Nach diesem Patch erzeugt die Backspace-Taste das Zeichen mit dem Code 127 und die Entfernen-Taste eine bekannte Escape-Sequenz.

Patchen Sie Kbd um einen Fehler in **setfont** zu beheben. Dieser Fehler tritt nur beim Kompilieren mit GCC-4.0.3 auf:

```
patch -Np1 -i ../kbd-1.12-gcc4_fixes-1.patch
```

Bereiten Sie Kbd zum Kompilieren vor:

```
./configure --datadir=/lib/kbd
```

Die Bedeutung der configure-Parameter:

`--datadir=/lib/kbd`
: Durch diesen Parameter werden die Daten zu Tastaturlayouts in einem Ordner abgelegt, der sich immer auf der root-Partition befindet, anstelle der Voreinstellung `/usr/share/kbd`.

Kompilieren Sie das Paket:

```
make
```

Dieses Paket enthält keine Testsuite.

Installieren Sie das Paket:

```
make install
```



Anmerkung

Für einige Sprachen (z. B. Belarussisch) hält Kbd keine nützliche Tastaturlayouttabelle vor, in der die Tabelle „by“ ISO-8859-5 annimmt, aber CP1251 verwendet wird. Benutzer solcher Sprachen sollten sich eine funktionierende Tastaturlayouttabelle herunterladen.

Einige der LFS-Bootskripte sing abhängig von den Kommandos **kbd_mode**, **openvt** und **setfont**. Da `/usr` in den früheren Phasen des Bootvorgangs noch nicht einghängt sein könnte, müssen sich diese Programm auf der root-Partition befinden:

```
mv -v /usr/bin/{kbd_mode,openvt,setfont} /bin
```

6.42.2. Inhalt von Kbd

Installierte Programme: chvt, deallocvt, dumpkeys, fgconsole, getkeycodes, kbd_mode, kbdrate, loadkeys, loadunimap, mapscrn, openvt, psfaddtable (Link auf psfxtable), psfgettable (Link auf psfxtable), psfstriptable (Link auf psfxtable), psfxtable, resizecons, setfont, setkeycodes, setleds, setmetamode, showconsolefont, showkey, unicode_start und unicode_stop

Kurze Beschreibungen

chvt	Ändert das aktive Virtuelle Terminal.
deallocvt	Gibt unbenutzte Virtuelle Terminals wieder frei.
dumpkeys	Gibt Tastaturübersetzungstabellen aus.
fgconsole	Gibt die Nummer des aktiven Virtuellen Terminals aus.
getkeycodes	Gibt die Scancode-zu-Keycode Zuweisungstabelle des Kernels aus.
kbd_mode	Setzt den Tastaturmodus bzw. zeigt ihn an.
kbdrate	Setzt die Tastenwiederholrate und -pausen oder zeigt sie an.
loadkeys	Lädt Tastaturübersetzungstabellen.
loadunimap	Lädt eine Unicode-zu-Schrift Zuweisungstabelle des Kernels.
mapscrn	Ein veraltetes Programm, das benutzerdefinierte Zeichenausgabe-Zuweisungstabellen in den Konsoletreiber lädt. Dies wird heutzutage durch **setfont** erledigt.
openvt	Startet ein Programm in einem neuen Virtuellen Terminal (VT).
psfaddtable	Ein Link auf **psfxtable**.
psfgettable	Ein Link auf **psfxtable**.
psfstriptable	Ein Link auf **psfxtable**.
psfxtable	Ein Satz von Werkzeugen zum Umgang mit Unicode-Zeichentabellen für Konsole-Schriften.
resizecons	Ändert die Vorstellung des Kernels über die Ausmaße einer Konsole.
setfont	Ändert EGA- (Enhanced Graphic Adapter) und VGA- (Video Graphics Array) Schriften in der Konsole.
setkeycodes	Lädt Scancode-zu-Keycode Zuweisungstabellen des Kernel. Nützlich, wenn Sie ein paar unübliche Tasten auf Ihrer Tastatur haben.
setleds	Stellt Tastaturoptionen und die LED's ein.
setmetamode	Definiert die Behandlung von Meta-Tasten auf der Tastatur.
showconsolefont	Zeigt die aktuelle EGA/VGA-Konsole-Schrift an.

showkey Zeigt Scancode, Keycode und ASCII-Code der auf der Tastatur gedrückten Taste an.

unicode_start Versetzt Tastatur und die Konsole in den UNICODE-Modus. Verwenden Sie dieses Programm nur, wenn Ihre Tastaturlayouttabelle eine ISO-8859-1-Kodierung verwendet. Mit anderen Kodierungen produziert es unbrauchbare Ergebnisse.

unicode_stop Schaltet den Unicode-Modus von Tastatur und Konsole wieder aus.

6.43. Less-394

Less ist ein Textanzeigeprogramm.

Geschätzte Kompilierzeit: 0.1 SBU
Ungefähr benötigter Festplattenplatz: 2.6 MB

6.43.1. Installation von Less

Bereiten Sie Less zum Kompilieren vor:

```
./configure --prefix=/usr --sysconfdir=/etc
```

Die Bedeutung der configure-Parameter:

--sysconfdir=/etc
Dieser Parameter bewirkt, dass die in diesem Paket installierten Programme ihre Konfigurationsdateien in `/etc` suchen.

Kompilieren Sie das Paket:

```
make
```

Dieses Paket enthält keine Testsuite.

Installieren Sie das Paket:

```
make install
```

6.43.2. Inhalt von Less

Installierte Programme: less, lessecho und lesskey

Kurze Beschreibungen

less Ein Dateibetrachter. Er zeigt den Inhalt einer Datei an und ermöglicht, darin zu blättern, nach Zeichenketten zu suchen und zu Markierungen springen.

lessecho Wird zum Expandieren von Metazeichen in Unix-Dateinamen benötigt (z. B. * und ?).

lesskey Wird zum Festlegen der Tastenbelegung für **less** verwendet.

6.44. Make-3.80

Das Paket Make enthält Werkzeuge zum Kompilieren von Software.

Geschätzte Kompilierzeit: 0.1 SBU
Ungefähr benötigter Festplattenplatz: 7.8 MB

6.44.1. Installation von Make

Bereiten Sie Make zum Kompilieren vor:

```
./configure --prefix=/usr
```

Kompilieren Sie das Paket:

```
make
```

Um das Ergebnis zu prüfen, führen Sie **make check** aus.

Installieren Sie das Paket:

```
make install
```

6.44.2. Inhalt von Make

Installiertes Programm: make

Kurze Beschreibungen

make Erkennt automatisch, welche Teile eines großen Programms (neu) kompiliert werden müssen und führt automatisch die notwendigen Kommandos aus.

6.45. Man-DB-2.4.3

Man-DB enthält Programme zum Finden und Anzeigen von Hilfeseiten (Man-pages).

Geschätzte Kompilierzeit: 0.2 SBU
Ungefähr benötigter Festplattenplatz: 9 MB

6.45.1. Installation von Man-DB

Zuerst nehmen Sie drei Anpassungen an den Quellen von Man-DB vor.

Die erste ändert den Standort der mitgelieferten übersetzten Hilfeseiten, damit diese sowohl im traditionellen als auch im UTF-8-Format verfügbar sind:

```
mv man/de{_DE.88591,} &&
mv man/es{_ES.88591,} &&
mv man/it{_IT.88591,} &&
mv man/ja{_JP.eucJP,} &&
sed -i 's,\*_\*,??,' man/Makefile.in
```

Die zweite Anpassung ist eine **sed**-Ersetzung, mit deren Hilfe die Zeile „/usr/man“ in `mandb.conf` auskommentiert wird. Dadurch werden redundante Ergebnisse vermieden, wenn Programme wie z. B. **whatis** verwendet werden:

```
sed -i '/\t\/usr\/man/d' src/man_db.conf.in
```

Die dritte Anpassung kümmert sich um Programme, die Man-DB zur Laufzeit finden sollte, aber derzeit noch nicht installiert sind:

```
cat >>include/manconfig.h.in <<"EOF"
#define WEB_BROWSER "exec /usr/bin/lynx"
#define COL "/usr/bin/col"
#define VGRIND "/usr/bin/vgrind"
#define GRAP "/usr/bin/grap"
EOF
```

Das Programm **col** ist ein Teil von Util-Linux, **lynx** ist ein textbasierter Web-Browser (siehe BLFS Installationsanleitung), **vgrind** wandelt Programmquellen in Groff-Eingaben um und **grap** ist nützlich für Typographiezeichen in Groff-Dokumenten. Normalerweise werden **vgrind** und **grap** zum Anzeigen von Handbuchseiten nicht benötigt. Sie sind weder Teil von LFS noch von BLFS, jedoch sollten Sie in der Lage sein, diese nach der Installation von LFS selbst zu installieren.

Bereiten Sie Man-DB zum Kompilieren vor:

```
./configure --prefix=/usr --enable-mb-groff --disable-setuid
```

Die Bedeutung der configure-Parameter:

--enable-mb-groff
: Dadurch verwendet **man** die Groff-Geräte „ascii8“ und „nippon“ zur Darstellung bzw. Formatierung von nicht-ISO-8859-1-kodierten Hilfeseiten.

--disable-setuid
Dadurch wird das Setuid-Bit auf dem Programm **man** für den Benutzer `man` deaktiviert.

Kompilieren Sie das Paket:

```
make
```

Dieses Paket enthält keine Testsuite.

Installieren Sie das Paket:

```
make install
```

Einige Pakete enthalten Man-Pages im UTF-8-Format, die von dieser Version von **man** nicht angezeigt werden können. Das folgende Skript macht es möglich, einige davon in die erwarteten Kodierungen der unteren Tabelle umzuwandeln. Man-DB erwartet die Hilfeseiten in einem Format wie unten gelistet und wandelt sie für die Darstellung automatische in die derzeitige Locale um. Auf diese Weise können die Hilfeseiten sowohl im traditionellen als auch im UTF-8-Format angezeit werden. Das Skript ist nur beschränkt einsetzbar beim Erstellen eines LFS-Systems und verzichtet daher auf Fehlerprüfung und unvorhersagbare Namen für temporäre Dateien.

```
cat >>convert-mans <<"EOF"
#!/bin/sh -e
FROM="$1"
TO="$2"
shift ; shift
while [ $# -gt 0 ]
do
        FILE="$1"
        shift
        iconv -f "$FROM" -t "$TO" "$FILE" >.tmp.iconv
        mv .tmp.iconv "$FILE"
done
EOF
install -m755 convert-mans  /usr/bin
```

Weitere Informationen zur Kompression von Man- und Info-pages erhalten Sie im BLFS-Buch unter *http://www.linuxfromscratch.org/blfs/view/cvs/postlfs/compressdoc.html*.

6.45.2. Nicht-Englische Hilfeseiten in LFS

Die Linux-Distributionen speichern die Hilfeseiten in unterschiedlichen Formaten ab. RedHat beispielsweise verwendet UTF-8, Debian setzt sprachspezifische 8-Bit-Kodierungen ein. Das für leider zu Inkompatibilitäten von Hilfeseiten, die für unterschiedliche Distributionen gedacht sind.

LFS setzt auf die gleichen Konventionen wie Debian. Das liegt daran, dass Man-DB nichts mit Hilfeseiten in UTF-8 anfangen kann. Des Weiteren ist Man-DB für unsere Zwecke dem herkömmlichen Man vorzuziehen, weil es ohne weitere Einrichtung für jede Locale funktioniert und nicht zuletzt, weil es derzeit keine voll funktionsfähige Implementierung der RedHat-Konvention gibt. RedHats **groff** formatiert bekanntermaßen Text falsch.

Weiter unten wird der Zusammenhang zwischen Sprachcodes und der erwarteten Kodierung einer Hilfeseite aufgelistet. Man-DB wandelt sie automatisch für die Darstellung in die richtige Locale um.

Tabelle 6.1. Erwartete Zeichenkodierung für Hilfeseiten

Sprache (Code)	**Kodierung**
Dänisch (da)	ISO-8859-1
Deutsch (de)	ISO-8859-1
Englisch (de)	ISO-8859-1
Spanisch (es)	ISO-8859-1
Finnisch (fi)	ISO-8859-1
Französisch (fr)	ISO-8859-1
Irisch (ga)	ISO-8859-1
Galician (gl)	ISO-8859-1
Indonesisch (id)	ISO-8859-1
Isländisch (is)	ISO-8859-1
Italienisch (it)	ISO-8859-1
Niederländisch (nl)	ISO-8859-1
Norwegisch (no)	ISO-8859-1
Portugiesisch (pt)	ISO-8859-1
Schwedisch (sv)	ISO-8859-1
Tchechisch (cs)	ISO-8859-2
Kroatisch (hr)	ISO-8859-2
Ungarisch (hu)	ISO-8859-2
Japanisch (ja)	EUC-JP
Koreanisch (ko)	EUC-KR
Polnisch (pl)	ISO-8859-2
Russisch (ru)	KOI8-R
Slovakisch (sk)	ISO-8859-2
Türkisch (tr)	ISO-8859-9



Anmerkung

Hilfeseiten in Sprachen, die sich nicht in der Tabelle befinden, werden nicht unterstützt. Norwegisch funktioniert aufgrund des Übergangs von no_NO zu nb_NO nicht und Koreanisch funktioniert aufgrund des unvollständigen Groff-Patches nicht.

Wenn ein Quellpaket die Hilfeseiten im erwarteten Format mitliefert, so können diese einfach nach `/usr/share/man/<Sprachcode>` kopiert werden. Beispielsweise können französische Hilfeseiten (*http://ccb.club.fr/man/man-fr-1.58.0.tar.bz2*) mit dem folgenden Kommando installiert werden:

```
mkdir -p /usr/share/man/fr &&
cp -rv man? /usr/share/man/fr
```

Falls die Programm-Entwickler die Hilfeseiten in UTF-8 ausliefern (z. B. „RedHat“) anstatt der oben aufgelisteten Kodierung, dann müssen sie vor der Installation von UTF-8 in die aufgelistete Kodierung umgewandelt werden. Dazu können Sie **convert-mans** verwenden. Die spanischen Hilfeseiten beispielsweise (*http://ditec.um.es/~piernas/manpages-es/man-pages-es-1.55.tar.bz2*) installieren Sie mit diesen Kommandos:

```
mv man7/iso_8859-7.7{,X}
convert-mans UTF-8 ISO-8859-1 man?/*.?
mv man7/iso_8859-7.7{X,}
make install
```



Anmerkung

`man7/iso_8859-7.7` muss von der Umwandlung ausgeschlossen werden, weil diese Datei bereits im Format ISO-8859-1 vorliegt (das ist ein Fehler bei der Paketierung in man-pages-es-1.55). Zukünftige Versionen benötigen diese Auslassung nicht.

6.45.3. Inhalt von Man-DB

Installierte Programme: accessdb, apropos, catman, convert-mans,lexgrog, man, mandb, manpath, whatis und zsoelim

Kurze Beschreibungen

accessdb Gibt den Inhalt der **whatis**-Datenbank in einer normal lesbaren Form aus.

apropos Durchsucht die **whatis**-Datenbank und gibt kurze Beschreibungen zu den Kommandos aus, die die angegebene Zeichenkette enthalten.

catman Erzeugt oder aktualisiert die vorformatierten Hilfeseiten.

convert-mans Formatiert Hilfeseite so um, dass Man-DB sie darstellen kann.

lexgrog Zeigt eine einzeilige Zusammenfassung über eine Hilfeseite an.

man Formatiert die angeforderte Hilfeseite und zeigt sie an.

mandb Erzeugt und aktualisiert **whatis**-Datenbanken.

manpath Zeigt den Inhalt von $MANPATH oder (falls $MANPATH nicht festgelegt ist) einen passenden Suchpfad basierend auf den Einstellungen in man.conf und der Umgebung des Benutzers an.

whatis Durchsucht die **whatis**-Datenbank und zeigt eine kurze Beschreibung zu den Systemkommandos an, die das übergebene Stichwort als separates Wort enthalten.

zsoelim Liest Dateien und ersetzt Zeilen der Form *.so* <*Datei*> > mit dem tatsächlichen Inhalt von <*Datei*>.

6.46. Mktemp-1.5

Das Paket Mktemp enthält Programme zum sicheren Anlegen temporärer Dateien aus Shell-Skripten heraus.

Geschätzte Kompilierzeit: weniger als 0.1 SBU
Ungefähr benötigter Festplattenplatz: 0.4 MB

6.46.1. Installation von Mktemp

Viele Skripte verwenden leider immer noch das missbilligte Programm **tempfile**, das die gleich Funktionalität hat wie **mktemp**. Patchen Sie mktemp, damit es auch einen Wrapper für **tempfile** installiert:

```
patch -Np1 -i ../mktemp-1.5-add_tempfile-3.patch
```

Bereiten Sie Mktemp zum Kompilieren vor:

```
./configure --prefix=/usr --with-libc
```

Die Bedeutung der configure-Parameter:

`--with-libc`
Dadurch benutzt **mktemp** die Funktionen *mkstemp* und *mkdtemp* aus der C-Bibliothek statt seiner eigenen Version.

Kompilieren Sie das Paket:

```
make
```

Dieses Paket enthält keine Testsuite.

Installieren Sie das Paket:

```
make install
make install-tempfile
```

6.46.2. Inhalt von Mktemp

Installierte Programme: mktemp und tempfile

Kurze Beschreibungen

mktemp Erzeugt temporäre Dateien auf sichere Weise. Es wird in Skripten verwendet.

tempfile Erzeugt temporäre Datein auf weniger sichere Weise als **mktemp**. Es wird aus Gründen der Rückwärtskompatibilität installiert.

6.47. Module-Init-Tools-3.2.2

Das Paket Module-Init-Tools enthält diverse Programme zur Verwaltung von Kernel-Modulen für Kernelversionen >= 2.5.47.

Geschätzte Kompilierzeit: weniger als 0.1 SBU
Ungefähr benötigter Festplattenplatz: 7 MB

6.47.1. Installation von Module-Init-Tools

Beheben Sie zuerst ein potentielles Problem, welches Auftritt wenn Module mittels regulärer Ausdrücke angegeben werden:

```
patch -Np1 -i ../module-init-tools-3.2.2-modprobe-1.patch
```

Führen Sie dieses Kommando aus, um die Testsuite zu starten (Anmerkung: **make distclean** ist nötig, um die Quelltexte aufzuräumen, denn Sie werden während dem Testlauf neu kompiliert):

```
./configure &&
make check &&
make distclean
```

Bereiten Sie Module-Init-Tools zum Kompilieren vor:

```
./configure --prefix=/ --enable-zlib
```

Kompilieren Sie das Paket:

```
make
```

Installieren Sie das Paket:

```
make INSTALL=install install
```

Die Bedeutung des make-Parameters:

INSTALL=install
: Normalerweise installiert **make install** die Binärdateien nicht, wenn sie bereits existieren. Durch diesen Parameter wird dieses Verhalten geändert und **install** statt dem sonst üblichen Skript aufgerufen.

6.47.2. Inhalt von Module-Init-Tools

Installierte Programme: depmod, generate-modprobe.conf, insmod, insmod.static, lsmod, modinfo, modprobe und rmmod

Kurze Beschreibungen

depmod	Erzeugt, basierend auf den Symbolen in existierenden Modulen, eine Abhängigkeitsdatei. Diese Datei wird von **modprobe** benutzt, um benötigte Module automatisch nachzuladen.
generate-modprobe.conf	Erzeugt die Datei modprobe.cond aus einer bestehenden Installation von 2.2er- oder 2.4er-Modulen.

insmod	Installiert ein ladbares Modul in den laufenden Kernel.
insmod.static	Eine statisch kompilierte Version von **insmod**.
lsmod	Listet die zur Zeit laufenden Kernelmodule auf.
modinfo	Untersucht eine mit einem Kernelmodul assoziierte Objektdatei und zeigt die darin verfügbaren Informationen an.
modprobe	Benutzt eine von **depmod** erzeugte Abhängigkeitsdatei, um benötigte Module automatisch nachzuladen.
rmmod	Entläd ein Modul aus dem laufenden Kernel.

6.48. Patch-2.5.4

Das Paket Patch enthält ein Programm zum Erzeugen oder Modifizieren von Dateien indem eine sogenannte „Patch"-Datei angewendet wird. Einen „Patch" erzeugt man üblicherweise mit **diff** und er beschreibt in maschinenlesbarer Form die Unterschiede zwischen zwei Versionen einer Datei.

Geschätzte Kompilierzeit: weniger als 0.1 SBU
Ungefähr benötigter Festplattenplatz: 1.6 MB

6.48.1. Installation von Patch

Bereiten Sie Patch zum Kompilieren vor.

```
./configure --prefix=/usr
```

Kompilieren Sie das Paket:

```
make
```

Dieses Paket enthält keine Testsuite.

Installieren Sie das Paket:

```
make install
```

6.48.2. Inhalt von Patch

Installiertes Programm: patch

Kurze Beschreibungen

patch Verändert Dateien nach den Vorgaben einer patch-Datei. Eine patch-Datei ist üblicherweise eine Auflistung von Unterschieden, die mit dem Programm **diff** erzeugt wurde. Durch Anwenden dieser Unterschiede auf die Originaldateien erstellt **patch** eine gepatchte Version.

6.49. Psmisc-22.2

Das Paket Psmisc enthält Programme zum Anzeigen von Prozessinformationen.

Geschätzte Kompilierzeit: weniger als 0.1 SBU
Ungefähr benötigter Festplattenplatz: 2.2 MB

6.49.1. Installation von Psmisc

Bereiten Sie Psmisc zum Kompilieren vor:

```
./configure --prefix=/usr --exec-prefix=""
```

Die Bedeutung der configure-Parameter:

--exec-prefix=""
: Dies stellt sicher, dass die Binärdateien von Psmisc nach `/bin` anstelle von `/usr/bin` installiert werden. Lt. FHS ist dies der korrekte Ort, weil einige der Programme in den LFS-Bootskripten verwendet werden.

Kompilieren Sie das Paket:

```
make
```

Dieses Paket enthält keine Testsuite.

Installieren Sie das Paket:

```
make install
```

pstree und **pstree.x11** müssen nicht in `/bin` liegen. Daher verschieben Sie sie nach `/usr/bin`:

```
mv -v /bin/pstree* /usr/bin
```

Normalerweise wird Psmisc's Programm **pidof** nicht installiert. Das ist meistens kein Problem weil wir später das Paket Sysvinit installieren, welches eine bessere Version von **pidof** installiert. Aber wenn Sie nicht Sysvinit verwenden möchten, können Sie die Installation von Psmisc durch Erstellen dieses Links komplettieren:

```
ln -sv killall /bin/pidof
```

6.49.2. Inhalt von Psmisc

Installierte Programme: fuser, killall, pstree und pstree.x11 (Link auf pstree)

Kurze Beschreibungen

fuser Zeigt die PIDs von Prozessen an, die gerade eine bestimmte Datei oder ein Dateisystem verwenden.

killall Beendet Prozesse aufgrund ihres Namens. Es sendet ein Signal an alle Prozesse, die ein bestimmtes Kommando ausführen.

oldfuser Zeigt die PIDs von Prozessen an, die gerade eine bestimmte Datei oder ein Dateisystem

verwenden.

pstree Zeigt laufende Prozesse als Baumstruktur an.

pstree.x11 Das gleiche wie **pstree**, wartet allerdings vor dem Beenden auf eine Bestätigung.

6.50. Shadow-4.0.15

Das Paket Shadow enthält Programme zur sicheren Verwaltung von Kennwörtern.

Geschätzte Kompilierzeit: 0.3 SBU
Ungefähr benötigter Festplattenplatz: 18.6 MB

6.50.1. Installation von Shadow



Anmerkung

Wenn Sie sichere Passwörter erzwingen möchten, sollten Sie vor der Installation von Shadow unter *http://www.linuxfromscratch.org/blfs/view/svn/postlfs/cracklib.html* nachlesen und Cracklib installieren. Fügen Sie dann den Parameter `--with-libcrack` zu dem folgenden **configure**-Kommando hinzu:

Bereiten Sie Shadow zum Kompilieren vor:

```
./configure --libdir=/lib --enable-shared --without-selinux
```

Die Bedeutung der configure-Parameter:

`--without-selinux`
Die Unterstützung für selinux ist in der Voreinstellung aktiviert, jedoch ist selinx nicht Teil eines Standard-LFS-Systems. Das **configure**-Skript würde ohne diesen Parameter mit einem Fehler abbrechen.

Verhindern Sie die Installation des Programmes **groups** und der zugehörigen Hilfeseite, da Coreutils eine bessere Version enthält:

```
sed -i 's/groups$(EXEEXT) //' src/Makefile
find man -name Makefile -exec sed -i '/groups/d' {} \;
```

Verhindern Sie die Installation der chinesischen und koreanischen Hilfeseiten, weil Man-DB sie nicht korrekt anzeigen kann:

```
sed -i -e 's/ ko//' -e 's/ zh_CN zh_TW//' man/Makefile
```

Shadow enthält weitere Hilfeseiten im UTF-8-Format. Man-DB kann diese in der empfohlenen Kodierung anzeigen, wenn Sie das Skript **convert-mans** verwenden, welches Sie erst kürzlich installiert haben.

```
 for i in de es fi fr id it pt_BR; do
    convert-mans UTF-8 ISO-8859-1 man/${i}/*.?
done

for i in cs hu pl; do
    convert-mans UTF-8 ISO-8859-2 man/${i}/*.?
done

convert-mans UTF-8 EUC-JP man/ja/*.?
convert-mans UTF-8 KOI8-R man/ru/*.?
convert-mans UTF-8 ISO-8859-9 man/tr/*.?
```

Kompilieren Sie das Paket:

```
make
```

Dieses Paket enthält keine Testsuite.

Installieren Sie das Paket:

```
make install
```

Shadow benutzt zwei Dateien zur Einrichtung der systemweiten Authentifizierungseinstellungen. Installieren Sie diese beiden Konfigurationsdateien:

```
cp -v etc/{limits,login.access} /etc
```

Sie sollten die voreingestellte *crypt*-Methode auf *MD5* ändern, welche sicherer ist. Außerdem ermöglicht sie Passwörter mit mehr als 8 Zeichen. Desweiteren müssen Sie den alten Standort der Benutzermailboxen von `/var/spool/mail` nach `/var/mail` ändern. Das erledigen Sie einfach, indem Sie die Konfigurationsdatei gleich beim Kopieren an die richtige Stelle ändern (benutzen Sie am besten „Kopieren und Einfügen“ um diesen Befehl auszuführen):

```
sed -e's@#MD5_CRYPT_ENAB.no@MD5_CRYPT_ENAB yes@' \
    -e 's@/var/spool/mail@/var/mail@' \
    etc/login.defs > /etc/login.defs
```



Anmerkung

Wenn Sie Shadow mit Unterstützung für Cracklib installieren, dann geben Sie das folgende **sed**-Kommando ein:

```
sed -i 's@DICTPATH.*@DICTPATH\t/lib/cracklib/pw_dict@' \
    /etc/login.defs
```

Verschieben Sie ein Programm an die korrekte Stelle:

```
mv -v /usr/bin/passwd /bin
```

Verschieben Sie Shadow's Bibliotheken an eine bessere Stelle:

```
mv -v /lib/libshadow.*a /usr/lib
rm -v /lib/libshadow.so
ln -sfv ../../lib/libshadow.so.0 /usr/lib/libshadow.so
```

Die Option *-D* zu **useradd** benötigt zur korrekten Funktion diesen Ordner:

```
mkdir -v /etc/default
```

6.50.2. Einrichten von Shadow

Dieses Paket enthält Werkzeuge zum Bearbeiten, Hinzufügen und Löschen von Benutzerpasswörtern. Wir werden hier nicht erläutern, was genau *password shadowing* bedeutet. Eine vollständige Erklärung finden Sie in der Datei `doc/HOWTO` in der entpackten Shadow-Ordnerstruktur. Eines gilt es allerdings zu beachten: Programme, die Passwörter überprüfen müssen (z. B. xdm, ftp und pop3 Server), müssen *shadow-konform* sein. Das heißt, sie müssen mit Shadow-Passwörtern umgehen können.

Um Shadow-Passwörter zu aktivieren, benutzen Sie dieses Kommando:

```
pwconv
```

Und um Shadow-Gruppenpasswörter zu aktivieren, benutzen Sie dieses Kommando:

```
grpconv
```

6.50.3. Vergeben des Passworts für root

Wählen Sie ein Kennwort für den Benutzer `root` und setzen Sie es mit dem Kommando:

```
passwd root
```

6.50.4. Inhalt von Shadow

Installierte Programme: chage, chfn, chgpasswd, chpasswd, chsh, expiry, faillog, gpasswd, groupadd, groupdel, groupmod, grpck, grpconv, grpunconv, lastlog, login, logoutd, newgrp, newusers, nologin, passwd, pwck, pwconv, pwunconv, sg (Link auf newgrp), su, useradd, userdel, usermod, vigr (Link auf vipw) und vipw
Installierte Bibliotheken: libshadow.{a,so}

Kurze Beschreibungen

chage Ändert die maximale Anzahl von Tagen zwischen zwei nötigen Passwortänderungen.

chfn Wird zum Ändern des vollständigen Namens und weiterer Informationen eines Benutzers benutzt.

chgpasswd Wird benutzt, um das Passwort mehrerer Gruppen in einem Durchlauf zu ändern.

chpasswd Wird benutzt, um das Passwort mehrerer Benutzer in einem Durchlauf zu ändern.

chsh Wird benutzt, um die voreingestellte Shell eines Benutzers zu ändern.

expiry Prüft, ob ein Kennwort abgelaufen ist und setzt eine entsprechende Regelung durch.

faillog Wird zum Untersuchen der Logdatei nach fehlgeschlagenen Logins, zum Setzen einer maximalen Fehlerzahl vor der Sperrung eines Kontos oder um den Zähler zurückzusetzen verwendet.

gpasswd Wird zum Hinzufügen und Löschen von Mitgliedern in Gruppen verwendet.

groupadd Erzeugt eine Gruppe mit dem angegebenen Namen.

groupdel Löscht eine Gruppe mit dem angegebenen Namen.

groupmod Ändert den Namen oder die GID einer Gruppe.

grpck Prüft die Integrität der Gruppen-Dateien `/etc/group` und `/etc/gshadow`.

grpconv Erzeugt oder aktualisiert die group-Datei von Shadow aus der normalen group-Datei.

grpunconv Aktualisiert `/etc/group` aus `/etc/gshadow` und löscht die letztere dann.

lastlog Berichtet über die letzten Anmeldungen aller oder eines bestimmten Benutzers.

login Wird vom System benutzt, um einen Benutzer anzumelden.

logoutd Ein Daemon, der Beschränkungen auf die Login-Zeit und -Ports durchsetzt.

newgrp Wird zum Ändern der aktuellen GID in einer Login-Sitzung benutzt.

newusers	Wird zum Erzeugen oder Aktualisieren einer Serie von Benutzerkonten in einem Durchlauf verwendet.
nologin	Zeigt einen Hinweis an, dass ein Benutzerkonto nicht verfügbar ist. Dies ist als Standard-Shell für deaktivierte Benutzerkonten gedacht.
passwd	Ändert das Passwort für einen Benutzer oder eine Gruppe.
pwck	Prüft die Integrität der Passwort-Dateien `/etc/passwd` und `/etc/shadow`.
pwconv	Erzeugt oder aktualisiert die Shadow-Passwort-Datei aus der normalen Passwort-Datei.
pwunconv	Aktualisiert `/etc/passwd` aus `/etc/shadow` und löscht letztere danach.
sg	Führt ein Kommando mit der angegebenen GID aus.
su	Führt eine Shell mit geänderter Benutzer- und Gruppen-ID aus.
useradd	Erzeugt einen neuen Benutzer mit dem angegebenen Namen oder aktualisiert die Vorgaben für neue Benutzer.
userdel	Löscht das angegebene Benutzerkonto.
usermod	Ändert Loginname, UID, Shell, Gruppe, Persönlichen Ordner und ähnliches für einen Benutzer.
vigr	Kann zum Bearbeiten von `/etc/group`- oder `/etc/gshadow`-Dateien benutzt werden.
vipw	Kann zum Bearbeiten von `/etc/passwd`- oder `/etc/shadow`-Dateien benutzt werden.
`libshadow`	Enthält Funktionen, die von den meisten der Programme in diesem Paket verwendet werden.

6.51. Sysklogd-1.4.1

Die in Sysklogd enthaltenen Programme dienen zum Aufzeichnen von Systemmeldungen, zum Beispiel denen des Kernels wenn ungewöhnliche Ereignisse auftreten.

Geschätzte Kompilierzeit: weniger als 0.1 SBU
Ungefähr benötigter Festplattenplatz: 0.6 MB

6.51.1. Installation von Sysklogd

Der folgende Patch behebt mehrere Probleme, unter anderem auch einen Kompilierfehler von Sysklogd mit Kerneln der 2.6er Serie:

```
patch -Np1 -i ../sysklogd-1.4.1-fixes-1.patch
```

Der folgende Patch sorgt dafür, dass sysklogd die Bytes im Bereich 0x80 bis 0x9f in einem Protokolleintrag literal gespeichert werden, anstatt sie durch ihre Oktalcodes zu ersetzen. Ohne diesen Patch würde sysklogd Protokolleinträge unleserlich machen:

```
patch -Np1 -i ../sysklogd-1.4.1-8bit-1.patch
```

Kompilieren Sie das Paket:

```
make
```

Dieses Paket enthält keine Testsuite.

Installieren Sie das Paket:

```
make install
```

6.51.2. Einrichtung von Sysklogd

Erstellen Sie nun die Konfigurationsdatei `/etc/syslog.conf`:

```
cat > /etc/syslog.conf << "EOF"
# Begin /etc/syslog.conf

auth,authpriv.* -/var/log/auth.log
*.*;auth,authpriv.none -/var/log/sys.log
daemon.* -/var/log/daemon.log
kern.* -/var/log/kern.log
mail.* -/var/log/mail.log
user.* -/var/log/user.log
*.emerg *

# End /etc/syslog.conf
EOF
```

6.51.3. Inhalt von Sysklogd

Installierte Programme: klogd und syslogd

Kurze Beschreibungen

klogd Ein System-Daemon zum Abfangen und Protokollieren von Kernel-Meldungen.

syslogd Protokolliert Meldungen, die von Systemprogrammen zum Protokollieren angeboten werden. Jede Meldung enthält zumindest einen Datumsstempel und den Hostnamen, und üblicherweise auch den Namen des Programms. Dies ist aber davon abhängig, wie vertrauensselig der Daemon eingestellt wurde.

6.52. Sysvinit-2.86

Das Sysvinit Paket enhält Programme, mit denen Sie das Starten, Ausführen und Beenden des Systems kontrollieren können.

Geschätzte Kompilierzeit: weniger als 0.1 SBU
Ungefähr benötigter Festplattenplatz: 1 MB

6.52.1. Installation von Sysvinit

Wenn Runlevel gewechselt werden (zum Beispiel beim Herunterfahren des Systems), sendet **init** Signale an alle Programme, die es gestartet hat. **Init** gibt „Sending processes the TERM signal“ auf dem Bildschirm aus. Dieser Text suggeriert, das **init** Signale an *alle* Prozesse sendet. Das ist so aber nicht korrekt, denn es geht hier nur um Prozesse, die von **init** gestartet wurden. Um diese Verwirrung zu vermeiden, können Sie die Quellen so modifizieren, dass es sich besser liest: „Sending processes started by init the TERM signal“:

```
sed -i 's@Sending processes@& started by init@g' \
    src/init.c
```

Kompilieren Sie das Paket:

```
make -C src
```

Dieses Paket enthält keine Testsuite.

Installieren Sie das Paket:

```
make -C src install
```

6.52.2. Einrichten von Sysvinit

Erstellen Sie die Datei `/etc/inittab`:

```
cat > /etc/inittab << "EOF"
# Begin /etc/inittab

id:3:initdefault:

si::sysinit:/etc/rc.d/init.d/rc sysinit

l0:0:wait:/etc/rc.d/init.d/rc 0
l1:S1:wait:/etc/rc.d/init.d/rc 1
l2:2:wait:/etc/rc.d/init.d/rc 2
l3:3:wait:/etc/rc.d/init.d/rc 3
l4:4:wait:/etc/rc.d/init.d/rc 4
l5:5:wait:/etc/rc.d/init.d/rc 5
l6:6:wait:/etc/rc.d/init.d/rc 6

ca:12345:ctrlaltdel:/sbin/shutdown -t1 -a -r now

su:S016:once:/sbin/sulogin

1:2345:respawn:/sbin/agetty tty1 9600
2:2345:respawn:/sbin/agetty tty2 9600
3:2345:respawn:/sbin/agetty tty3 9600
```

```
4:2345:respawn:/sbin/agetty tty4 9600
5:2345:respawn:/sbin/agetty tty5 9600
6:2345:respawn:/sbin/agetty tty6 9600

# End /etc/inittab
EOF
```

6.52.3. Inhalt von Sysvinit

Installierte Programme: bootlogd, halt, init, killall5, last, lastb (Link auf last), mesg, mountpoint, pidof (Link auf killall5), poweroff (Link auf halt), reboot (Link auf halt), runlevel, shutdown, sulogin, telinit (Link auf init), utmpdump und wall

Kurze Beschreibungen

bootlogd Protokolliert Bootmeldungen in eine Datei.

halt Ruft üblicherweise **shutdown** mit dem Parameter *-h* auf, außer wenn der aktuelle Runlevel 0 ist, dann teilt es dem Kernel mit, das System anzuhalten. Vorher vermerkt es in `/var/log/wtmp`, dass das System nun heruntergefahren wird.

init Der erste gestartete Prozess nachdem der Kernel die Hardware initialisiert hat. **Init** übernimmt den Bootvorgang und startet alle anstehenden Programme.

killall5 Sendet ein Signal an alle Prozesse, außer denen in der eigenen Sitzung—so beendet es nicht die Programme, die das Skript ausführen welches es aufgerufen hat.

last Zeigt, welcher Benutzer als letztes eingeloggt und ausgeloggt hat, indem es die Datei `/var/log/wtmp` durchsucht. Es kann auch Systemstarts und -stopps sowie Wechsel der Runlevel zeigen.

lastb Zeigt die letzten fehlgeschlagenen Login-Versuche, die in `/var/log/btmp` protokolliert wurden.

mesg Kontrolliert, welche anderen Benutzer Nachrichten auf das aktuelle Terminal senden können.

mountpoint Prüft, ob der Ordner ein Mountpunkt ist.

pidof Gibt die PIDs eines Programms aus.

poweroff Weist den Kernel an, das System anzuhalten und den Computer auszuschalten. Siehe auch die Beschreibung zu **halt**.

reboot Weist den Kernel an, das System neu zu starten. Siehe auch die Beschreibung zu **halt**.

runlevel Zeigt den vorigen und den aktuellen Runlevel an. Die nötigen Informationen werden aus `/var/run/utmp` gelesen.

shutdown Fährt das System sicher herunter, sendet entsprechende Signale an alle Prozesse und benachrichtig alle angemeldeten Benutzer.

sulogin Ermöglicht `root` sich einzuloggen. Dies wird normalerweise von **init** gestartet, wenn das System im Einbenutzermodus gestartet wurde.

telinit Weist **init** an, in den angegebenen Runlevel zu wechseln.

utmpdump Zeigt den Inhalt der angegebenen Logindatei in einem benutzerfreundlicheren Format an.

wall Schreibt eine Nachricht an alle angemeldeten Benutzer.

6.53. Tar-1.15.1

Das Paket Tar enthält ein Archivprogramm.

Geschätzte Kompilierzeit: 0.2 SBU
Ungefähr benötigter Festplattenplatz: 13.7 MB

6.53.1. Installation von Tar

Wenden Sie diesen Patch an, um ein paar Fehler in der Testsuite unter Verwendung von GCC-4.0.3 zu beheben:

```
patch -Np1 -i ../tar-1.15.1-gcc4_fix_tests-1.patch
```

Tar hat bei Dateien größer 4GB ein Problem mit dem Parameter *-S*. Der folgende Patch behebt das Problem:

```
patch -Np1 -i ../tar-1.15.1-sparse_fix-1.patch
```

Einige neuer Versionen von Tar sind anfällig für einen Pufferüberlauf in speziell dafür manipulierten Tar-Archiven. Der folgende Patch behebt das Problem:

```
patch -Np1 -i ../tar-1.15.1-security_fixes-1.patch
```

Bereiten Sie Tar zum Kompilieren vor:

```
./configure --prefix=/usr --bindir=/bin --libexecdir=/usr/sbin
```

Kompilieren Sie das Paket:

```
make
```

Um das Ergebnis zu prüfen, führen Sie **make check** aus.

Installieren Sie das Paket:

```
make install
```

6.53.2. Inhalt von Tar

Installierte Programme: rmt und tar

Kurze Beschreibungen

rmt Mit diesem Programm kan man ein magnetorientiertes Bandlaufwerk an einem entfernten Rechner steuern. Zur Kommunikation wird Interprozesskommunikation verwendet.

tar Wird zum Erzeugen, Auflisten und Extrahieren von Dateien aus einem Archiv verwendet. Diese Archive werden oft auch als „Tarball“ bezeichnet.

6.54. Texinfo-4.8

Das Paket Texinfo enthält Programme zum Lesen, Schreiben und Konvertieren von Info-Seiten (Systemdokumentation).

Geschätzte Kompilierzeit: 0.2 SBU
Ungefähr benötigter Festplattenplatz: 16.6 MB

6.54.1. Installation von Texinfo

Das Programm **info** geht davon aus, dass ein Text dieselbe Anzahl Zeichen auf dem Bildschirm wie Bytes im Speicher verbraucht und das ein solcher Text an jeder beliebigen Stelle getrennt werden kann. Dies schlägt in UTF-8-basierten Locales natürlich fehl. Der folgende Patch umgeht das Problem, indem auf englischsprachige Meldungen zurückgegriffen wird, wenn eine Multibyte-Locale verwendet wird.

```
patch -Np1 -i ../texinfo-4.8-multibyte-1.patch
```

Texinfo ermöglicht es lokalen Benutzern durch eine sog. Symlink-Attacke bestimmte Dateien zu überschreiben. Der folgende Patch behebt das Problem:

```
patch -Np1 -i ../texinfo-4.8-tempfile_fix-2.patch
```

Bereiten Sie Texinfo zum Kompilieren vor:

```
./configure --prefix=/usr
```

Kompilieren Sie das Paket:

```
make
```

Um das Ergebnis zu prüfen, führen Sie **`make check`** aus.

Installieren Sie das Paket:

```
make install
```

Optional können Sie auch die zu einer typischen TeX-Installation gehörenden Pakete installieren:

```
make TEXMF=/usr/share/texmf install-tex
```

Die Bedeutung des make-Parameters:

`TEXMF=/usr/share/texmf`
Die Makefile-Variable `TEXMF` enthält den Pfad zu Ihrem TeX Basisordner, falls später TeX installiert wird.

Das Info-Dokumentationssystem speichert die Liste der Menüeinträge in einer einfachen Textdatei. Die Datei liegt in `/usr/share/info/dir`. Unglücklicherweise können die Einträge in dieser Datei durch Probleme mit Makefile-Dateien einzelner Pakete durcheinander geraten. Falls Sie diese Datei jemals neu erzeugen müssen, ist Ihnen das folgende Kommando dabei behilflich:

```
cd /usr/share/info
rm dir
for f in *
do install-info $f dir 2>/dev/null
done
```

6.54.2. Inhalt von Texinfo

Installierte Programme: info, infokey, install-info, makeinfo, texi2dvi, texi2pdf und texindex

Kurze Beschreibungen

info	Wird zum Lesen von Info-Dokumenten benutzt. Info-Dokumente sind Man-pages sehr ähnlich, gehen aber oft tiefer in die Materie als einfach nur die möglichen Parameter zu beschreiben. Vergleichen Sie beispielsweise **man bison** und **info bison**.
infokey	Kompiliert eine Quelldatei mit Info-Anpassungen in ein binäres Format.
install-info	Wird zum Installieren von Info-Dateien benutzt. Es aktualisiert die Einträge in der **info**-Indexdatei.
makeinfo	Übersetzt Texinfo Quelldokumente in verschiedene andere Formate: Info-Dateien, reiner Text, oder HTML.
texi2dvi	Wird zum Formatieren von Texinfo-Dokumenten in ein Geräteunabhängiges Format zum Drucken benutzt.
texi2pdf	Wird zum Konvertieren von Texinfo-Dokumenten in das portable Document Format (PDF) verwendet.
texindex	Sortiert Texinfo-Indexdateien.

6.55. Udev-096

Das Paket Udev enthält Programme zum dynamischen Erzeugen von Gerätedateien.

Geschätzte Kompilierzeit: 0.1 SBU
Ungefähr benötigter Festplattenplatz: 6.8 MB

6.55.1. Installation von Udev

Das Archiv udev-config enthält LFS-spezifische Konfigurationsdateien für Udev. Entpacken Sie das Archiv in den Quellordner von Udev:

```
tar xf ../udev-config-6.2.tar.bz2
```

Erzeugen Sie einige Geräte und Ordner die Udev nicht bereitstellen kann, weil sie sehr früh während dem Bootvorgang benötigt werden:

```
install -dv /lib/{firmware,udev/devices/{pts,shm}}
mknod -m0666 /lib/udev/devices/null c 1 3
ln -sv /proc/self/fd /lib/udev/devices/fd
ln -sv /proc/self/fd/0 /lib/udev/devices/stdin
ln -sv /proc/self/fd/1 /lib/udev/devices/stdout
ln -sv /proc/self/fd/2 /lib/udev/devices/stderr
ln -sv /proc/kcore /lib/udev/devices/core
```

Kompilieren Sie das Paket:

```
make EXTRAS="extras/ata_id extras/cdrom_id extras/edd_id \
            extras/firmware extras/floppy extras/path_id \
            extras/scsi_id extras/usb_id extras/volume_id"
```

Die Bedeutung der make-Option:

EXTRAS=...
: Dadurch werden einige Hilfsprogramme erzeugt, die beim Erstellen von eigenen Udev-Regeln behilflich sind.

Zum Testen der Ergebnisse führen Sie dieses Kommando aus: **`make test`**.

Hinweis: Die Udev-Testsuite wird zahlreiche Einträge im Systemlog verursachen. Diese Meldungen sind alle harmlos und können einfach ignoriert werden.

Installieren Sie das Paket:

```
make DESTDIR=/ \
    EXTRAS="extras/ata_id extras/cdrom_id extras/edd_id \
            extras/firmware extras/floppy extras/path_id \
            extras/scsi_id extras/usb_id extras/volume_id" install
```

Die Bedeutung des make-Parameters:

DESTDIR=/
: Dies verhindert, dass die Installationsroutine von Udev jegliche Instanzen von **udevd** beendet, die möglicherweise auf dem Host-System laufen.

Udev muss vor der ersten Verwendung eingerichtet werden, weil die Installationsroutine keinerleit Konfigurationsdateien installiert. Installieren Sie also nun die LFS-spezifischen Konfigurationsdateien:

```
cp -v udev-config-6.2/[0-9]* /etc/udev/rules.d/
```

Installieren Sie die Dokumentation. Sie erklärt unter anderem, wie man eigene Udev-Regeln schreibt:

```
install -m644 -D -v docs/writing_udev_rules/index.html \
    /usr/share/doc/udev-096/index.html
```

6.55.2. Inhalt von Udev

Installierte Programme: ata_id, cdrom_id, create_floppy_devices, edd_id, firmware_helper, path_id, scsi_id, udevcontrol, udevd, udevinfo, udevmonitor, udevsettle, udevtest, udevtrigger, usb_id, vol_id und write_cd_aliases
Installierter Ordner: /etc/udev

Kurze Beschreibungen

ata_id	Stellt Udev eine einmalige Beschreibung und weitere Informationen (uuid, label) für ein ATA-Laufwerk zur Verfügung.
cdrom_id	Stellt Udev die Geräteeigenschaften von CD-ROM- und DVD-ROM-Laufwerken zur Verfügung.
create_floppy_devices	Erstellt alle möglichen Diskettenlaufwerks-Gerätedateien basierend auf dem CMOS-Typ.
edd_id	Stellt Udev die EDD-ID für ein BIOS-Laufwerk zur Verfügung.
firmware_helper	Lädt Firmware in angeschlossene Geräte.
path_id	Stellt den kürzesten einmaligen Pfad zu einer Hardware zur Verfügung.
scsi_id	Stellt Udev einen einmaligen SCSI-Bezeichner zur Verfügung. Dieser basiert auf dem Rückgabewert einer SCSI INQUIRY-Anfrage an das angegebene Gerät.
udevcontrol	Stellt einige Parameter zum Ausführen des **udevd**-Daemon ein. Dazu gehört z. B. die Protokollierstufe.
udevd	Dieser Daemon wacht über uevents an einem netlink-Socket, erzeugt Geräte-Dateien und führt bestimmte externe Programme als Reaktion auf diese uevents aus.
udevinfo	Ermöglicht Anwendern, die Udev-Datenbank nach Informationen über die zur Zeit verfügbare Geräte im System abzufragen. Es stellt außerdem eine Möglichkeit dar, jedes Gerät im `sysfs`-Dateisystem abzufragen, um beim Erzeugen von udev-Regeln behilflich zu sein.

udevmonitor Zeigt die vom Kernel erhaltenen Ereignisse und die von Udev erzeugte Reaktion darauf an, nachdem eine Regel abgearbeitet wurde.

udevsettle Überwacht die Warteschlange der Udev-Ereignisse und beendet sich, wenn alle wartenden Ereignisse abgearbeitet wurden.

udevtest Simuliert ein Udev-Ereignis für das angegebene Gerät und gibt den Namen der Gerätedatei oder der Netzwerkschnittstelle aus, die ein echter **udev**-Aufruf für dieses Gerät erzeugt hätte.

udevtrigger Sorgt für eine Wiederholung der Kernel-Geräte-Ereignisse.

usb_id Stellt Udev Informationen zu USB-Geräten zur Verfügung.

vol_id Stellt Udev label und uuid eines Dateisystems zur Verfügung.

`/etc/udev` Enthält Udev-Konfigurationsdateien, Geräteberechtigungen und Regeln für die Namensvergabe von **udev**.

6.56. Util-linux-2.12r

Das Paket Util-linux enthält verschiedene Werkzeuge. Darunter befinden sich Programme zum Umgang mit Dateisystemen, Konsolen, Partitionen und (System-)Meldungen.

Geschätzte Kompilierzeit: 0.2 SBU
Ungefähr benötigter Festplattenplatz: 17.2 MB

6.56.1. Anmerkung zur FHS-Konformität

FHS empfiehlt, `/var/lib/hwclock` anstelle des eigentlich üblichen Ordners `/etc` als Speicherort für die Datei `adjtime` zu benutzen. Führen Sie das folgende Kommando aus, um das Programm **hwclock** FHS-Konform zu machen:

```
sed -i 's@etc/adjtime@var/lib/hwclock/adjtime@g' \
    hwclock/hwclock.c
mkdir -p /var/lib/hwclock
```

6.56.2. Installation von Util-linux

Util-linux lässt sich mit neueren Versionen der Linux-Libc-Header nicht kompilieren. Der folgende Patch behebt das Problem:

```
patch -Np1 -i ../util-linux-2.12r-cramfs-1.patch
```

Bereiten Sie Util-linux zum Kompilieren vor:

```
./configure
```

Kompilieren Sie das Paket:

```
make HAVE_KILL=yes HAVE_SLN=yes
```

Die Bedeutung der make-Parameter:

HAVE_KILL=yes
: Verhindert, dass das Programm **kill** (bereits durch Procps installiert) erneut kompiliert und installiert wird.

HAVE_SLN=yes
: Verhindert, dass das Programm **sln** (eine statisch gelinkte Version von **ln**, bereits durch Glibc installiert) erneut kompiliert und installiert wird.

Dieses Paket enthält keine Testsuite.

Installieren Sie das Paket:

```
make HAVE_KILL=yes HAVE_SLN=yes install
```

6.56.3. Inhalt von Util-linux

Installierte Programme: agetty, arch, blockdev, cal, cfdisk, chkdupexe, col, colcrt, colrm, column, ctrlaltdel, cytune, ddate, dmesg, elvtune, fdformat, fdisk, flock, fsck.cramfs, fsck.minix, getopt, hexdump, hwclock, ipcrm, ipcs, isosize, line, logger, look, losetup, mcookie, mkfs, mkfs.bfs, mkfs.cramfs, mkfs.minix, mkswap, more, mount, namei, pg, pivot_root, ramsize (Link auf rdev), raw, rdev, readprofile, rename, renice, rev, rootflags (Link auf rdev), script, setfdprm, setsid, setterm, sfdisk, swapoff (Link auf swapon), swapon, tailf, tunelp, ul, umount, vidmode (Link auf rdev), whereis und write

Kurze Beschreibungen

agetty	Öffnet einen tty-Port, fragt nach dem Login-Namen und startet das Programm **login**.
arch	Gibt die Systemarchitektur aus.
blockdev	Ermöglicht den Aufruf von Blockgeräte-ioctls an der Kommandozeile.
cal	Zeigt einen einfachen Kalender an.
cfdisk	Wird zum Bearbeiten der Partitionstabelle eines Gerätes benutzt.
chkdupexe	Findet Duplikate von ausführbaren Dateien.
col	Filtert Rückwärts-Zeilenvorschübe aus.
colcrt	Filtert **nroff**-Ausgaben für Terminals, denen bestimmte Fähigkeiten fehlen, wie zum beispiel durchstreichen oder halbe Zeilen.
colrm	Filtert eine bestimmte Spalte aus.
column	Formatiert eine Datei in mehrere Spalten.
ctrlaltdel	Setzt die Funktion der Tastenkombination Strg-Alt-Entf auf einen Hart- oder Softreset.
cytune	Wurde benutzt, um die Parameter der seriellen Schnittstellen auf Cyclade-Karten zu verändern.
ddate	Gibt das Diskordianische Datum aus, oder konvertiert ein Gregorianisches Datum in ein Diskordianisches.
dmesg	Zeigt die Bootmeldungen des Kernel an.
elvtune	Kann zum Manipulieren der Performance und Interaktivität von Blockgeräten benutzt werden.
fdformat	Formatiert eine Diskette low-level.
flock	Beansprucht eine Dateisperrung und führt während der Sperrung ein Kommando aus.
fdisk	Wird zum Bearbeiten der Partitionstabelle eines Gerätes benutzt.
fsck.cramfs	Führt eine Konsistenzprüfung auf einem Cramfs-Dateisystem durch.
fsck.minix	Führt eine Konsistenzprüfung auf einem Minix-Dateisystem durch.
getopt	Analysiert die Optionen in der Kommandozeile.
hexdump	Zeigt eine Datei hexadezimal oder in einem anderen Format an.
hwclock	Wird zum Setzen oder Lesen der Hardware-Uhr (auch RTC- oder BIOS-Uhr gennant) benutzt.

ipcrm Entfernt die angegebene IPC-Ressource (Inter-Process Communication).

ipcs Gibt IPC Status-Informationen aus.

isosize Gibt die Größe eines iso9660-Dateisystems aus.

line Kopiert eine einzelne Zeile.

logger Gibt eine Nachricht in das Logsystem ein.

look Sucht nach Zeilen, die mit einer bestimmten Zeichenkette beginnen, und zeigt sie an.

losetup Konfiguriert und kontrolliert Loopback-Geräte.

mcookie Erzeugt magische Cookies (hexadezimale 128-bit Zufallszahlen) für xauth.

mkfs Erzeugt ein Dateisystem auf einem Gerät (üblicherweise einer Festplattenpartition).

mkfs.bfs Erzeugt ein SCO-bfs-Dateisystem (Santa Cruz Operations).

mkfs.cramfs Erzeugt ein cramfs-Dateisystem.

mkfs.minix Erzeugt ein Minix-Dateisystem.

mkswap Initialisiert ein Gerät oder eine Datei als Auslagerungsbereich.

more Ein Filter zum seitenweisen Anzeigen von Text. Less ist jedoch besser.

mount Hängt das auf dem Gerät vorhandene Dateisystem in einem Ordner ein.

namei Zeigt die symbolischen Links in Pfadnamen an.

pg Zeigt eine Textdatei seitenweise an.

pivot_root Macht ein Dateisystem zu dem neuen root-Dateisystem für den aktuellen Prozess.

ramsize Kann zum Setzen der Größe einer RAM-Disk in einem bootbaren Abbild benutzt werden.

raw Bindet ein zeichenorientiertes Linux-raw-Gerät an ein Blockgerät.

rdev Kann in einem bootfähigen Abbild das root-Gerät abfragen und festlegen.

readprofile Liest Profiling-Informationen aus dem Kernel.

rename Benennt eine Datei um und ersetzt ein Zeichenkette durch eine andere.

renice Verändert die Priorität eines Prozesses.

rev Dreht die Zeilen einer Datei um.

rootflags Kann die root-Parameter eines bootfähigen Abbildes festlegen.

script Erstellt eine Abschrift einer Terminalsitzung.

setfdprm Setzt benutzerdefinierte Floppy-Disk-Parameter.

setsid Führt ein Kommando in einer neuen Sitzung aus.

setterm Stellt Terminal-Attribute ein.

sfdisk Kann Festplattenpartitionen bearbeiten.

swapoff Deaktiviert Auslagerungsdateien und -geräte.

swapon Aktiviert Auslagerungsdateien und -geräte und zeigt bereits verwendete Geräte und Dateien an.

tailf Verfolgt das Wachstum einer Protokolldatei. Zeigt zuerst die letzten zehn Zeilen einer Protokolldatei an und hängt dann der Reihe nach neu hinzugekommene Zeilen an die Ausgabe an.

tunelp Justiert Parameter eines Zeilendruckers.

ul Ein Filter zum Übersetzen von Unterstrichen in entsprechende Escape-Sequenzen, die das verwendete Terminal versteht.

umount Löst ein Dateisystem aus der Ordnerstruktur.

vidmode Kann zum Setzen des Videomodus in einem bootfähigen Abbild benutzt werden.

whereis Gibt den Ort der Binärdatei, der Quellen und der Man-pages für ein Kommando aus.

write Sendet eine Nachricht an einen Benutzer (sofern der Benutzer den Empfang solcher Nachrichten nicht deaktiviert hat).

6.57. Vim-7.0

Das Paket Vim enthält einen sehr mächtigen Texteditor.

Geschätzte Kompilierzeit: 0.4 SBU
Ungefähr benötigter Festplattenplatz: 47.4 MB



Alternativen zu Vim

Wenn Sie einen anderen Editor bevorzugen—zum Beispiel Emacs, Joe oder Nano—dann schauen Sie unter *http://www.linuxfromscratch.org/blfs/view/svn/postlfs/editors.html*, dort finden Sie einige Installationshinweise.

6.57.1. Installation von Vim

Entpacken Sie zuerst beide Archivdateien—`vim-7.0.tar.bz2` und (optional) `vim-7.0-lang.tar.gz`— in den gleichen Ordner. Dann behebn Sie mit dem folgenden Patch einige Fehler, die von den Entwicklern seit der letzten veröffentlichten Version von Vim-7.0 gefunden haben:

```
patch -Np1 -i ../vim-7.0-fixes-7.patch
```

Diese Version von Vim installiert einige übersetzte Hilfeseiten und kopiert sie in einen Ordner, der nicht von Man-DB durchsucht wird. Patchen Sie Vim deshalb so, dass die Hilfeseiten in durchsuchte Ordner installiert werden und in das gewünschte Format umgewandelt werden können:

```
patch -Np1 -i ../vim-7.0-mandir-1.patch
```

Durch einen der Upstream-Patches gibt es ein Problem mit dem Herunterladen der Sprachdateien mittels HTTP. Solange die Entwickler dieses Problem nicht behoben haben, hilft dieser Patch weiter:

```
patch -Np1 -i ../vim-7.0-spellfile-1.patch
```

Ändern Sie schlussendlich noch den Speicherort für die Konfigurationsdatei `vimrc` nach `/etc`:

```
echo '#define SYS_VIMRC_FILE "/etc/vimrc"' >> src/feature.h
```

Bereiten Sie Vim zum Kompilieren vor:

```
./configure --prefix=/usr --enable-multibyte
```

Die Bedeutung der configure-Parameter:

`--enable-multibyte`
: Dieser Parameter schaltet die Unterstützung zum Editieren von Dateien mit Multibyte-Zeichenkodierung ein. Das wird benötigt, wenn Sie ein Locale mit Multibyte-Zeichensatz verwenden. Dieser Parameter ist auch hilfreich, wenn Sie Dateien bearbeiten möchten, die mit Distributionen wie z. B. Fedora Core erzeugt wurden (diese Distribution benutzt UTF-8 als voreingestellen Zeichensatz).

Kompilieren Sie das Paket:

```
make
```

Wenn Sie das Ergebnis testen möchten, können Sie dazu **`make test`** verwenden. Die Testsuite gibt jedoch

eine Menge sinnlose Zeichen auf dem Bildschirm aus und könnte die Einstellungen Ihres Terminals durcheinander bringen. Sie können die Ausgabe in eine Datei umleiten, um dieses Problem zu umgehen.

Installieren Sie das Paket:

```
make install
```

In UTF-8-Locales versucht **vimtutor**, die Anleitungen von ISO-8859-1 nach UTF-8 umzuwandeln. Einige liegen aber gar nicht in ISO-8859-1 vor und werden durch diese Umwandlung unleserlich. Falls Sie `vim-7.0-lang.tar.gz` installiert haben und ein UTF-8-Locale einsetzen werden, so entfernen Sie bitte die nicht-ISO-8859-1-Anleitungen. Stattdessen wird dann eine englische Version benutzt.

```
rm -f /usr/share/vim/vim70/tutor/tutor.{gr,pl,ru,sk}
rm -f /usr/share/vim/vim70/tutor/tutor.??.*
```

Viele Benutzer sind es gewöhnt, **vi** anstelle von **vim** zu starten. Damit **vim** gestartet wird, obwohl **vi** eingegeben wurde, erzeugen Sie einen symbolischen Link sowohl für die Binärdatei als auch für die Hilfeseite in den verfügbaren Sprachen:

```
ln -sv vim /usr/bin/vi
for L in "" fr it pl ru; do
    ln -sv vim.1 /usr/share/man/$L/man1/vi.1
done
```

In der Voreinstellung wird die Dokumentation zu Vim in `/usr/share/vim` installiert. Durch den folgenden symbolischen Link wird sie unter `/usr/share/doc/vim-7.0` verfügbar und ist damit konsistent mit der Dokumentation anderer Pakete:

```
ln -sv ../vim/vim70/doc /usr/share/doc/vim-7.0
```

Falls Sie später ein X Window-System auf Ihrem LFS installieren möchten, sollten Sie nach der Installation von X Ihren Vim erneut installieren. Vim bringt eine grafische Oberfläche mit, die allerdings X und ein paar weitere Bibliotheken voraussetzt. Weitere Informationen finden Sie in der Dokumentation zu Vim und im BLFS-Buch unter *http://www.linuxfromscratch.org/blfs/view/svn/postlfs/editors.html#postlfs-editors-vim*.

6.57.2. Einrichten von Vim

In der Voreinstellung läuft **vim** im vi-inkompatiblen Modus. Das ist wahrscheinlich neu für Leute, die in der Vergangenheit andere Editoren verwendet haben. Die Einstellung „nocompatible“ ist dennoch unten aufgeführt, um daran zu erinnern, dass das neue Verhalten benutzt wird. Wenn Sie zum vi-kompatiblen Modus wechseln möchten, sollte „compatible“ im Kopfbereich der Datei stehen. Das ist nötig, weil diese Option viele Voreinstellungen für Parameter vornimmt. Ihre eigenen Änderungen an diesen Parametern müssen danach erfolgen, weil sie sonst von „compatible“ zurückgesetzt würden. Erzeugen Sie eine Standard vim-Konfigurationsdatei mit diesem Kommando:

```
cat > /etc/vimrc << "EOF"
" Begin /etc/vimrc

set nocompatible
set backspace=2
syntax on
if (&term == "iterm") || (&term == "putty")
  set background=dark
endif

" End /etc/vimrc
EOF
```

Der Parameter `set nocompatible` versetzt **vim** in einen nützlicheren Betriebsmodus (Voreinstellung) als den vi-kompatiblen Modus. Entfernen Sie das „no“ falls Sie das alte **vi**-Verhalten nutzen möchten. *set backspace=2* erlaubt das sogenannte Backspacing über Zeilenumbrüche hinweg, automatisches Einrücken und das Starten von Einrückungen. *syntax on* aktiviert **vim**s Hervorheben von Syntax. Schließlich stellt die *if*-Verzweigung sicher, dass mittels `set background=dark` die Hintergrundfarbe von bestimmten Terminals besser eingestellt ist. Dadurch wird hervorgehobene Syntax in diesen Terminal-Emulatoren besser lesbar.

Die Dokumentation zu weiteren möglichen Optionen erhalten Sie mit diesem Kommando:

```
vim -c ':options'
```



Anmerkung

Normalerweise installiert Vim die Dateien zur Rechtschreibprüfung nur in englischer Sprache. Wenn Sie die Rechtschreibprüfung auch für Ihre Sprache verfügbar haben möchten, laden Sie bitte die `*.spl`- und optional auch die `*.sug`-Dateien für Ihre Sprache und Kodierung von *ftp://ftp.vim.org/pub/vim/runtime/spell/* herunter und speichern Sie sie nach `/usr/share/vim/vim70/spell/`.

Um diese Sprachdateien zu verwenden, müssen Sie in `/etc/vimrc` eingerichtet werden. Beispiel:

```
set spelllang=en,ru
set spell
```

Weitere Informationen finden Sie in der Datei README unter der gleichen Adresse.

6.57.3. Inhalt von Vim

Installierte Programme: efm_filter.pl, efm_perl.pl, ex (Link auf vim), less.sh, mve.awk, pltags.pl, ref, rview (Link auf vim), rvim (Link auf vim), shtags.pl, tcltags, vi (Link auf vim), view (Link auf vim), vim, vim132, vim2html.pl, vimdiff (Link auf vim), vimm, vimspell.sh, vimtutor und xxd

Kurze Beschreibungen

efm_filter.pl	Ein Filter zum Erzeugen einer Fehlerdatei, die von **vim** gelesen werden kann.
efm_perl.pl	Reformatiert Fehlermeldungen von Perl, um sie mit dem Quickfix-Modus von „vim“ benutzen zu können.
ex	Startet **vim** im ex-Modus.
less.sh	Ein Skript, welches **vim** mit less.vim startet.
mve.awk	Verarbeitet **vim**-Fehler.
pltags.pl	Erzeugt eine Markup-Datei für Perl-Code, die mit **vim** benutzt werden kann.
ref	Prüft die Schreibweise von Argumenten.
rview	Eine eingeschränkte Version von **view**: es gibt keine Shell-Kommandos und **view** kann nicht angehalten werden.
rvim	Eine eingeschränkte Version von **vim**: es gibt keine Shell-Kommandos und **vim** kann nicht angehalten werden.

shtags.pl	Erzeugt eine Markup-Datei für Perl-Skripte.
tcltags	Erzeugt eine Markup-Datei für TCL-Code.
view	Startet **vim** im Nur-lesen-Modus.
vi	Link auf **vim**.
vim	Dies ist der Editor.
vim132	Startet **vim** in einem Terminal mit 132-Spalten-Modus.
vim2html.pl	Konvertiert Vim-Dokumentation zu HyperText Markup Language (HTML).
vimdiff	Editiert zwei oder drei Versionen einer Datei mit **vim** und zeigt die Unterschiede an.
vimm	Aktiviert das DEC Locator-Eingabemodell auf einem entfernten Terminal.
vimspell.sh	Untersucht eine Datei und erzeugt die nötigen Syntax-Regeln um das Hervorheben der Syntax in **vim** zu ermöglichen. Dieses Skript benötigt das alte Unix-Kommando **spell**, welches allerdings weder von LFS, noch von BLFS bereitgestellt wird.
vimtutor	Bringt Ihnen die wichtigsten Tastenbelegungen und Kommandos von **vim** bei.
xxd	Erzeugt eine Hex-Ausgabe einer Datei. Das geht auch umgekehrt und kann zum Patchen von Binärdateien benutzt werden.

6.58. Informationen zu Debugging Symbolen

Die meisten Programme und Bibliotheken werden in der Voreinstellung mit Debugging-Symbolen kompiliert (mit der Option **gcc** `-g`). Wenn Sie ein Programm oder eine Bibliothek debuggen, die mit debugging Symbolen kompiliert wurde, kann Ihnen der Debugger nicht nur die Speicheradressen, sondern auch die Namen der Funktionen und der Variablen im Programm anzeigen.

Doch das Einbinden dieser Debugging-Symbole vergrößert das Programm bzw. die Bibliothek deutlich. Das folgende Beispiel soll Ihnen einen Eindruck über den von Debugging-Symbolen benötigten Speicher geben:

- Eine **bash**-Binärdatei mit Debugging-Symbolen: 1200 KB
- Eine **bash**-Binärdatei ohne Debugging-Symbole: 480 KB
- Glibc und GCC-Dateien (`/lib` und `/usr/lib`) mit Debugging-Symbolen: 87 MB
- Glibc und GCC-Dateien ohne Debugging-Symbole: 16 MB

Die Größen variieren ein wenig, abhängig vom Compiler und der eingesetzten C-Bibliothek. Aber wenn man Programme mit und ohne Debugging-Symbole vergleicht, liegt der Faktor normalerweise zwischen 2 und 5.

Vermutlich werden Sie niemals einen Debugger mit Ihrer Systemsoftware einsetzen, daher können Sie durch das Entfernen der Symbole eine Menge Platz sparen. Der Einfachheit halber finden Sie im nächsten Kapitel ein Kommando, mit dem Sie alle debugging Symbole von allen Programmen und Bibliotheken auf Ihrem System entfernen können. Weitere Informationen zum Thema Optimierung finden Sie in der Anleitung unter *http://www.linuxfromscratch.org/hints/downloads/files/optimization.txt*.

6.59. Erneutes Stripping

Da Sie Ihre Systemsoftware vermutlich nicht debuggen möchten, können Sie hier ca. 90MB Platz sparen. Dazu entfernen Sie die Debugging-Symbole. Das zieht keine Probleme nach sich, aber Sie können die verkleinerten Programme danach nicht mehr vollständig debuggen.

Normalerweise gibt es mit dem folgenden Kommando keine Schwierigkeiten. Aber Sie könnten z. B. einen Tippfehler machen und dadurch das System unbrauchbar machen. Bevor Sie **strip** ausführen, sollten Sie ein Backup machen.

Wenn Sie strip ausführen möchten, ist besondere Vorsicht geboten, damit Sie strip nicht auf Programme anwenden, die gerade ausgeführt werden—inklusive der Bash-Shell. Daher müssen Sie die chroot-Umgebung vorerst verlassen:

```
logout
```

Und dann erneut betreten:

```
chroot $LFS /tools/bin/env -i \
    HOME=/root TERM=$TERM PS1='\u:\w\$ ' \
    PATH=/bin:/usr/bin:/sbin:/usr/sbin \
    /tools/bin/bash --login
```

Nun können die Debugging-Symbole sicher aus Binärdateien und Bibliotheken entfernt werden:

```
/tools/bin/find /{,usr/}{bin,lib,sbin} -type f \
  -exec /tools/bin/strip --strip-debug '{}' ';'
```

Es werden viele Dateien gemeldet, deren Format nicht erkannt wurde. Die meisten dieser Dateien sind Skripte und keine Binärdateien. Die Warnungen können einfach ignoriert werden.

Wenn Sie sehr wenig Platz auf der Festplatte haben, können Sie `--strip-all` auf die Binärdateien in `/{,usr/}{bin,sbin}` anwenden und so nochmals mehrere Megabytes sparen. Benutzen Sie diese Option jedoch nicht mit Bibliotheken—sie würden zerstört werden.

6.60. Aufräumen

Von nun an müssen Sie das folgende Kommando zum Betreten der chroot-Umgebung verwenden:

```
chroot "$LFS" /usr/bin/env -i \
    HOME=/root TERM="$TERM" PS1='\u:\w\$ ' \
    PATH=/bin:/usr/bin:/sbin:/usr/sbin \
    /bin/bash --login
```

Der Grund dafür ist, dass Sie keine Programme mehr aus `/tools` benötigen. Sie können den Ordner nun löschen.



Anmerkung

Wenn Sie `/tools` löschen, werden auch die temporären Kopien von Tcl, Expect und DejaGNU gelöscht (die Sie zum Testen der Toolchain benutzt haben). Wenn Sie diese Programme später noch benutzen möchten, müssen Sie sie neu kompilieren und installieren. Im BLFS-Buch finden Sie die entsprechenden Anleitungen dafür (siehe auch *http://www.linuxfromscratch.org/blfs/*).

Falls die Einbindung der virtuellen Kernel-Dateisysteme verloren gegangen ist (z. B. durch Entmounten oder einen Neustart), so müssen Sie diese vor dem Betreten der chroot-Umgebung erneut einbinden. Die Vorgehensweise ist unter Abschnitt 6.2.2, „Einhängen und Füllen von /dev“ und Abschnitt 6.2.3, „Einhängen der virtuellen Kernel-Dateisysteme“ erklärt.

Kapitel 7. Aufsetzen der System-Bootskripte

7.1. Einführung

In diesem Kapitel werden Sie die LFS-Bootskripte aufsetzen. Die meisten Skripte funktionieren ohne Anpassungen, aber ein paar benötigen eine Konfigurationsdatei weil sie beispielsweise mit Hardware an Ihrem Computer zu tun haben.

LFS verwendet Bootskripte im sehr gebräuchlichen System-V-Stil. Es gibt auch andere Möglichkeiten. Beispielsweise finden Sie unter *http://www.linuxfromscratch.org/hints/downloads/files/bsd-init.txt* eine Anleitung für BSD-Init. Oder durchsuchen Sie die LFS-Mailinglisten nach „depinit“ um eine andere Variante zu versuchen.

Falls Sie sich für etwas ganz anderes entscheiden sollten, können Sie dieses Kapitel ganz überspringen und direkt bei Kapitel 8 fortfahren.

7.2. LFS-Bootskripte-6.2

Das Paket LFS-Bootskripte enthält die Skripte zum Starten und Stoppen des Systems beim Booten und Herunterfahren Ihres Computers.

Geschätzte Kompilierzeit: weniger als 0.1 SBU
Ungefähr benötigter Festplattenplatz: 0.4 MB

7.2.1. Installation von LFS-Bootskripte

Installieren Sie das Paket:

```
make install
```

7.2.2. Inhalt von LFS-Bootskripte

Installierte Skripte: checkfs, cleanfs, console, functions, halt, ifdown, ifup, localnet, mountfs, mountkernfs, network, rc, reboot, sendsignals, setclock, static, swap, sysklogd, template und udev

Kurze Beschreibungen

checkfs Prüft die Integrität von Dateisystemen bevor sie eingehängt werden (mit der Ausnahme von journal- und netzwerkbasierten Dateisystemen).

cleanfs Entfernt Dateien, die nicht über einen Neustart hinaus existieren sollten. Dazu gehören zum Beispiel die Dateien in `/var/run/` und `/var/lock/`. Es erzeugt `/var/run/utmp` und entfernt die eventuell vorhandenen Dateien `/etc/nologin`, `/fastboot` und `/forcefsck`.

console Läd das für Ihre Tastatur korrekte Tastaturlayout und stellt die Bildschirmschriftart ein.

functions Enthält allgemeine Funktionen die von verschiedenen Skripten genutzt werden. Dazu gehören z. B. Fehler- oder Statusprüfung.

halt Hält das System an.

ifdown Unterstützt das Netzwerkskript beim Stoppen von Netzwerkgeräten.

ifup Unterstützt das Netzwerkskript beim Starten von Netzwerkgeräten.

localnet Setzt den Hostnamen und das lokale Loopback-Gerät auf.

mountfs Hängt alle nicht als *noauto* markierten und nicht netzwerkbasierten Dateisysteme ein.

mountkernfs Hängt virtuelle Kernel-basierte Dateisysteme ein (z. B. `proc`).

network Macht Netzwerkschnittstellen wie z. B. Netzwerkkarten verfügbar und richtet — wenn nötig — das Standard-Gatway ein.

rc Das Haupt-Runlevel-Kontrollskript. Es ist dafür verantwortlich, alle anderen Skripte eins nach dem anderen in der richtigen Reihenfolge auszuführen.

reboot Startet das System neu.

sendsignals Stellt sicher, dass jeder Prozess beendet wird, bevor das System herunterfährt oder neu startet.

setclock Setzt die Kernelzeit auf lokale Zeit, falls die Hardware-Uhr nicht auf UTC-Zeit eingestellt ist.

static Stellt Funktionen zum Zuweisen einer statischen IP-Adresse an ein Netzwerkgerät zur Verfügung.

swap Aktiviert und deaktiviert Swap-Dateien und -Partitionen.

sysklogd Startet und stoppt die System- und Kernel-Log-Daemons.

template Eine Vorlage, die Sie verwenden können, um Ihre eigenen Bootskripte für eigene Daemons zu schreiben.

udev Bereitet `/dev` vor und startet Udev.

7.3. Wie funktionieren diese Bootskripte?

Linux benutzt eine spezielle Bootmethode mit dem Namen SysVinit. Sie basiert auf dem Konzept der *Runlevel*. Dieses Konzept kann in verschiedenen Distributionen sehr unterschiedlich umgesetzt sein. Nehmen Sie also nicht an, nur weil etwas in Distribution XY funktioniert, geht es in LFS auf die gleiche Weise. LFS respektiert zwar allgemein übliche Standards, geht aber dennoch (wie alle anderen) seinen eigenen Weg.

SysVinit (wir nennen es nun einfach nur „init“) funktioniert nach dem Konzept der Runlevel. Es gibt 7 Runlevel (von 0 bis 6), genaugenommen gibt es sogar noch mehr, aber diese sind für Spezialfälle reserviert und werden üblicherweise nicht benutzt. `init(8)` beschreibt diese Details genauer. Jeder Runlevel korrespondiert mit Skripten oder Diensten, die der Computer beim Hochfahren ausführen bzw. starten oder stoppen soll. Der Standard-Runlevel ist 3. Hier sehen Sie eine Übersicht, wie die Runlevel üblicherweise eingesetzt werden:

0: Fährt den Computer herunter
1: Ein-Benutzer-Modus
2: Mehr-Benutzer-Modus ohne Netzwerk
3: Mehr-Benutzer-Modus mit Netzwerk
4: reserviert für eigene Anpassungen, funktioniert ansonsten wie 3
5: genauso wie 4, wird normalerweise für grafischen Login benutzt (wie z. B. X's **xdm** oder KDE's **kdm**)
6: Startet den Computer neu

Das Kommando zum Wechseln des Runlevel ist **init *`<Runlevel>`***, wobei *`<Runlevel>`* den Runlevel angibt, in den Sie wechseln möchten. Zum Neustarten des Computers würde ein Benutzer zum Beispiel **init 6** eingeben. Das **reboot**-Kommando ist nur ein Alias darauf, genauso wie das Kommando **halt** ein Alias auf **init 0** ist.

Unter `/etc/rc.d` befinden sich eine Menge Ordner mit dem Namen `rc?.d`, wobei das ? die Nummer eines Runlevels ist. Dort liegt auch der Ordner `rcsysinit.d`, er enthält einige symbolische Links. Einige beginnen mit einem *K*, andere mit einem *S*, gefolgt von einer zweistelligen Zahl. Das K bedeutet beenden (*k*ill) eines Dienstes, das S bedeutet starten (*s*tart) eines Dienstes. Die Zahlen bestimmen die Reihenfolge, in der die Skripte ausgeführt werden und können zwischen 00 und 99 liegen. Je kleiner die Zahl, desto früher wird das Skript ausgeführt. Wenn init in einen anderen Runlevel wechselt, werden die nötigen Skripte gestoppt und andere dafür gestartet.

Bisher war nur von Links die Rede. Die echten Skripte befinden sich in `/etc/rc.d/init.d`. Sie erledigen die eigentliche Arbeit, denn die ganzen symbolischen Links zeigen nur auf sie. Stopp- und Startskripte zeigen jeweils auf dieselbe Datei in `/etc/rc.d/init.d`. Das funktioniert, weil die Bootskripte mit unterschiedlichen Parametern aufgerufen werden können: zum Beispiel *`start`*, *`stop`*, *`restart`*, *`reload`*, *`status`*. Wenn ein K-Link ausgeführt werden soll, wird das entsprechende Skript mit dem *`stop`*-Parameter aufgerufen. Wenn ein S-Link ausgeführt werden soll, wird das Skript mit dem *`start`*-Parameter aufgerufen.

Es gibt eine Ausnahme: *S*-Links in den Ordnern `rc0.d` und `rc6.d` starten keine Dienste. Sie werden stattdessen mit dem Parameter *`stop`* aufgerufen um etwas zu beenden. Die Grund dafür ist, dass Sie wohl kaum einen Dienst starten möchten, wenn Sie rebooten oder das System herunterfahren.

Hier die Beschreibungen, welche Parameter zu einem Skript was bewirken:

`start`
: Der Dienst wird gestartet.

`stop`

Der Dienst wird gestoppt.

restart
Der Dienst wird gestoppt und dann erneut gestartet.

reload
Die Konfiguration des Dienstes wird neu eingelesen. Das verwendet man, nachdem die Konfigurationsdatei eines Dienstes geändert wurde und man nicht den ganzen Dienst neu starten muss.

status
Gibt aus, ob der Dienst läuft, und wenn ja, mit welchen PIDs.

Sie können den Bootprozess natürlich nach Ihren Wünschen anpassen (schlussendlich ist es ja Ihr eigenes Linux). Die Dateien hier sind nur Beispiele dafür, wie man es gut erledigen kann.

7.4. Umgang mit Geräten und Modulen an einem LFS-System

In Kapitel 6 haben Sie Udev installiert. Bevor wir zu den Details kommen wie das alles funktioniert, möchten wir Ihnen erst einen Rückblick darüber geben, wie man früher mit Geräten unter Linux umgegangen ist.

Traditionell hat man unter Linux eine statische Methode zum Erzeugen von Gerätedateien benutzt. Dabei wurden sehr viele Gerätedateien vorab in `/dev` erzeugt (manchmal mehrere tausend). Dabei war es völlig egal, ob die zugehörige Hardware tatsächlich existierte oder nicht. Dies wurde typischerweise durch das Skript **MAKEDEV** erledigt, welches eine Menge Systemaufrufe mit dem Programm **mknod** und den entsprechenden Gerätenummern durchführte und so Gerätedateien zu allen erdenklichen Geräten erzeugte.

Mit der Udev-Methode werden nur die Gerätedateien erzeugt, zu denen der Kernel auch ein Gerät gefunden hat. Weil diese Gerätedateien bei jedem Systemstart neu erzeugt werden, speichert man sie auf einem sog. `tmpfs`-Dateisystem. Dieses Dateisystem existiert nur im Arbeitsspeicher und verbraucht daher keinen Festplattenplatz. Gerätedateien benötigen kaum Platz, auf diese Weise wird also nur sehr wenig Arbeitsspeicher verbaucht.

7.4.1. Die Entwicklungsgeschichte von Udev

Im Februar 2000 wurde ein neues Dateisystem mit dem Namen `devfs` in den Kernel 2.3.46 integriert und dann in der 2.4er Serie der stabilen Kernel verfügbar gemacht. Obwohl es in den Kernelquellen selbst verfügbar war, hat diese Methode nie wirkliche Unterstützung von den Kernel-Entwicklern bekommen.

Das Haupt-Problem bei diesem von `devfs` adaptierten Ansatz war die Art und Weise, auf die Geräte erkannt, erzeugt und benannt wurden. Letzteres (Namensvergabe) war wohl das kritischste Problem. Das Dateisystem `devfs` litt außerdem unter sog. Race conditions die mit dem Konzept zusammenhingen und nicht ohne nennenswerte Änderungen am Kernel geändert werden konnten. Außerdem wurde es als „missbilligt“ markiert, weil es nicht mehr gepflegt wurde.

Mit der Enticklung der 2.5er Entwickler-Kernelserie, die später als stabile 2.6er-Serie veröffentlicht wurde, wurde ein neues Dateisystem mit dem Namen `sysfs` eingeführt. Die Aufgabe von `sysfs` ist es, die Betriebssystemstruktur an Anwenderprozesse zu exportieren. Mit dieser aus der Anwenderschicht sichtbaren Repräsentation des Betriebssystems kam ein Ersatz für `devfs` in Sichtweite.

7.4.2. Udev-Implementierung

7.4.2.1. Sysfs

Das Dateisystem `sysfs` wurde oben schon kurz erwähnt. Man fragt sich vielleicht, woher `sysfs` von den Geräten und den zu verwendenden Gerätenummern weiß: Treiber, die direkt in den Kernel integriert wurden, registrieren sich bei `sysfs` sobald sie vom Kernel erkannt werden. Bei Kernel-Modulen geschieht dieser Vorgang beim Laden des Moduls. Sobald `sysfs` in das System eingehängt ist (unter `/sys`), sind die Daten von den mit `sysfs` registrierten Treibern für Prozesse aus der Anwenderschicht, und damit auch für **udev**, verfügbar.

7.4.2.2. Das Udev-Bootskript

Das Bootskript **S10udev** kümmert sich um das Erstellen von Gerätedateien beim Systemstart. Das Skript entfernt **/sbin/hotplug** als Verantwortliches Skript für uevents, weil der Kernel kein externes Programm mehr benötigt. Stattdessen wartet **udevd** an einem Netlink-Socket auf uevents des Kernels. Als nächstes kopiert das Bootskript statische Gerätedateien von `/lib/udev/devices` nach `/dev`. Dies ist wichtig, weil einige Gerätedateien, Ordner und symbolische Links beim Bootvorgang benötigt werden, bevor die dynamische Geräteerstellung von Udev betriebsbereit ist. Durch Einrichten von statischen Gerätedateien in `/lib/udev/devices` kann man auch Gerätedateien unterstützen, die normalerweise nicht von Udev automatisch angelegt werden würden. Als nächstes startet das Bootskript den Udev-Daemon **udevd**, der von nun an auf uevents wartet und reagiert. Schlussendlich zwingt das Bootskript den Kernel, die uevents für Geräte zu wiederholen, die sich vor dem Start von **udevd** registriert haben.

7.4.2.3. Erzeugen von Gerätedateien

Udev verlässt sich auf die Informationen von `sysfs` in `/sys` und liest daraus die Haupt- und Unterkennung für Gerätedateien aus. Beispielsweise enthält `/sys/class/tty/vcs/dev` den Text „7:0“. Diesen Wert interpretiert **udevd** und erzeugt eine Gerätedatei mit der Hauptkennung *7* und der Unterkennung *0*. Die Namen und Berechtigungen für die in `/dev` erzeugten Gerätedateien ergeben sich aus den definierten Regeln in `/etc/udev/rules.d/`. Die dort abgelegten Regeln sind ähnlich nummeriert wie die Dateien der LFS-Bootskripte. Falls **udevd** keine Regel für ein erzeugtes Gerät auffinden kann, ist die Voreinstellung für die Berechtigungen *660* und die Gerätedatei gehört *root:root*. Eine genauere Dokumentation zu den Konfigurationsdateien von Udev finden Sie unter `/usr/share/doc/udev-096/index.html`.

7.4.2.4. Laden von Modulen

Als Modul kompilierte Gerätetreiber können Aliase eingebaut haben. Diese kann man sich mit dem Kommando **modinfo** ansehen und hängen üblicherweise mit den Bus-Spezifischen Kennmarken eines vom Treiber unterstützten Gerätes zusammen. Beispielsweise unterstützt der Treiber *snd-fm801* PCI-Geräte mit der Hersteller-ID 0x1319 und Geräte-ID 0x0801. Der zugehörige Alias lautet „pci:v00001319d00000801sv*sd*bc04sc01i*“. Für die meisten Geräte exportiert der Bus-Treiber den Alias des notwendigen Treibers nach `sysfs`. So würde beispielsweise die Datei `/sys/bus/pci/devices/0000:00:0d.0/modalias` den Wert „pci:v00001319d00000801sv00001319sd00001319bc04sc01i00“ enthalten. Die mit LFS installierten Udev-Regeln sorgen dafür, dass **udevd /sbin/modprobe** mit dem Inhalt der uevent-Umgebungsvariable `MODALIAS` aufruft (sie sollte das Gleiche enthalten wie die Datei `modalias` in sysfs). Dadurch werden alle Module aufgerufen, deren Alias dem Wert entsprechen.

In diesem Beispiel bedeutet das aber auch, dass zusätzlich zu *snd-fm801* noch er veraltete (und unerwünschte) Treiber *forte* geladen wird, sofern er verfügbar ist. Weiter unten ist eine Möglichkeit beschrieben, wie man das Laden unerwünschter Treiber verhindern kann.

Der Kernel selbst ist ebenfalls in der Lage, Module für Netzwerkprotokolle, Dateisystem und NLS nach Bedarf zu laden.

7.4.2.5. Der Umgang mit dynamischen bzw. Hotplug-Geräten

Wenn Sie ein Gerät wie beispielsweise einen USB-MP3-Player, so erkennt der Kernel ein neu angeschlossenes Gerät und erzeugt einen uevent. Um dieses neue uevent kümmert sich dann **udevd** so wie oben beschrieben.

7.4.3. Probleme mit dem Laden von Kernelmodulen und dem Erzeugen von Gerätedateien

Es gibt ein paar mögliche Probleme beim automatisierten Erzeugen von Gerätedateien:

7.4.3.1. Das nötige Kernelmodul wird nicht automatisch geladen

Udev lädt nur dann ein Modul, wenn ein Bus-Spezifischer Alias vorhanden ist und der Treiber die nötigen Aliase korrekt nach `sysfs` exportiert. Wenn dies nicht der Fall ist, muss man sich auf andere Weise um das Laden des Moduls kümmern. Mit Linux-2.6.16.27 kann Udev korrekt programmierte Treiber für INPUT-, IDE-, PCI-, USB-, SCSI-, SERIO- und FireWire-Geräte laden.

Mit dem Kommando **modinfo** und dem Modulnamen als Argument können Sie herausfinden, ob der von Ihnen benötigte Treiber von Udev unterstützt wird. Versuchen Sie nun, den Geräte-Ordner unter `/sys/bus` zu finden und prüfen Sie die dortige Datei `modalias`.

Wenn die Datei `modalias` unter `sysfs` vorhanden ist und der Treiber das Gerät unterstützt, aber der Alias fehlt, so ist dies ein Fehler im Treiber. Dann müssen Sie den Treiber ohne Hilfe von Udev laden und darauf hoffen, dass dieser Fehler später behoben wird.

Wenn die Datei `modalias` in dem zugehörigen Ordner unter `/sys/bus` nicht vorhanden ist, so haben die Kernel-Entwickler für diesen Bus-Typ noch keine Modalias-Unterstützung programmiert. Bei Linux-2.6.16.27 ist dies z. B. der Fall für den ISA-Bus. Dies wird wahrscheinlich in einer zukünftigen Kernelversion behoben.

Udev sorgt sich nicht um das Laden sogenannter „wrapper"-Treibern wie beispielsweise *snd-pcm-oss* oder Nicht-Hardware-Treibern wie *loop*.

7.4.3.2. Ein Kernelmodul lädt nicht automatisch und Udev ist nicht dafür zuständig

Wenn ein „Wrapper"-Modul nur die Funktionen eines anderen Moduls erweitert (so erweitert z. B. das Modul *snd-pcm-oss* die Funktionalität von *snd-pcm* indem es die Soundkarte auch OSS-Anwendungen zur Verfügung stellt), dann richten Sie **modprobe** so ein, dass es das Wrapper-Modul lädt, nachdem Udev das Hauptmodul geladen hat. Dies erreichen Sie mit einer „install"-Anweisung in `/etc/modprobe.conf`. Beispiel:

```
install snd-pcm /sbin/modprobe -i snd-pcm ; \
    /sbin/modprobe snd-pcm-oss ; true
```

Wenn es sich bei dem fraglichen Modul nicht um einen Wrapper handelt sondern alleinstehend geladen wird, so richten Sie bitte das Bootskript **S05modules** ein, sodass das Modul beim Booten geladen wird. Dies erreichen Sie, indem Sie den Modulnamen an die Datei `/etc/sysconfig/modules` in einer eigenen Zeile anhängen. Dies funktioniert natürlich auch mit Wrapper-Modulen, ist aber nicht optimal.

7.4.3.3. Udev lädt unerwünschte Module

Entweder Sie kompilieren das fragliche Modul gar nicht erst, oder Sie schließen es mit Hilfe der schwarzen Liste in `/etc/modprobe.conf` aus, so wie mit dem Modul *forte* im folgenden Beispiel:

```
blacklist forte
```

Module auf der schwarzen Liste können natürlich weiterhin von Hand mit dem Programm **modprobe** geladen werden.

7.4.3.4. Udev erzeugt eine Gerätedatei falsch oder setzt einen falschen symbolischen Link

Dies geschieht für gewöhnlich, wenn eine Regel versehentlich auf ein anderes Gerät passt, als vorgesehen. Eine schlecht geschriebene Regel könnte z. B. sowohl auf eine SCSI-Festplatte (wie gewünscht) als auch auf das zugehörige generische SCSI-Gerät (unerwünscht) nach dem Hersteller passen. Sie müssen die Regel auffinden und genauer ausformulieren.

7.4.3.5. Udev funktioniert nur unzuverlässig

Dies ist zumeist nur ein weiteres Symptom des zuvor beschriebenen Problems. Falls nicht, und die betreffende Regel `sysfs`-Attribute verwendet, so könnte es sich um Kernel-Zeitprobleme handeln, die erst in zukünftigen Kernelversionen behoben werden. Sie können das Problem umgehen, indem Sie eine Regel schreiben, die erst auf das verwendete `sysfs`-Attribut wartet und fügen Sie an `/etc/udev/rules.d/10-wait_for_sysfs.rules` an. Wenn Sie dies tun, informieren Sie bitte das LFS-Entwicklerteam darüber und teilen Sie uns auch mit, ob dies funktioniert.

7.4.3.6. Udev erzeugt eine Gerätedatei nicht

Im folgenden Text wird davon ausgegangen, dass der Treiber entweder statisch in den Kernel eingebaut ist, oder das Modul bereits geladen ist. Außerdem sollten Sie überprüft haben, ob nicht möglicherweise nur eine Gerätedatei mit falschen Namen erzeugt wurde.

Udev hat nicht genügen Informationen zum Erzeugen einer Gerätedatei, wenn der Kerneltreiber seine Daten nicht ins `sysfs` exportiert. Dies ist bei Treibern von Drittherstellern außerhalb des Kernelbaums leider öfter der Fall. Erzeugen Sie eine statische Gerätedatei mit der korrekten Haupt- und Unterkennung in `/lib/udev/devices`. Ziehen Sie dazu auch die Datei `devices.txt` aus der Kernel-Dokumentation zu Rate oder lesen Sie die Dokumentation des Drittherstellers. Diese statische Gerätedatei wird dann beim Bootvorgang von **S10udev** nach `/dev` kopiert.

7.4.3.7. Die Reihenfolge der Gerätenamen ändert sich mit jedem Bootvorgang

Dies liegt daran, dass Udev (gewollt und ganz bewusst) alle uevents parallel—und somit in unterschiedlicher Reihenfolge—abarbeitet. Dieses Phänomen wird niemals „repariert“ werden. Verlassen Sie sich nicht auf die Gerätenamen des Kernels. Schreiben Sie stattdessen Regeln, die aussagekräftige symbolische Links mit stabilen Namen erzeugen. Dazu können Sie Attribute zu Geräten heranziehen, wie z B. Seriennummern oder die Ausgabe der verschiedenen *_id-Hilfsprogramme von Udev. Schauen Sie für einige Beispiele unter Abschnitt 7.12, „Erzeugen von benutzerdefinierten symbolischen Links zu Geräten“Abschnitt 7.13, „Einrichten des network-Skripts“ nach.

7.4.4. Nützliche Dokumentation

Weitere hilfreiche Dokumentation finden Sie an den folgenden Stellen:

- A Userspace Implementation of `devfs` *http://www.kroah.com/linux/talks/ ols_2003_udev_paper/Reprint-Kro*
- udev FAQ *http://www.kernel.org/pub/linux/utils/kernel/hotplug/udev-FAQ*
- The `sysfs` Filesystem *http://www.kernel.org/pub/linux/kernel/people/mochel/doc/papers/ols-2005/mochel.pd*

7.5. Einrichten des setclock-Skripts

Das Skript **setclock** liest die Zeit aus der Hardware-Uhr des Computers (auch bekannt als BIOS- oder CMOS-Uhr) und konvertiert sie mit Hilfe von `/etc/localtime` (falls die Hardware Uhr auf GMT gestellt ist) in lokale Zeit. Die Datei `/etc/localtime` enthält die Information, in welcher Zeitzone sich der Anwender befindet. Wenn die Hardware-Uhr auf lokale Zeit eingestellt ist, wird die Zeit nicht konvertiert. Es gibt leider keinen Weg um automatisch herauszufinden, ob die Hardware-Uhr auf GMT gestellt ist oder nicht, deshalb müssen Sie diese Einstellung selber vornehmen.

Falls Sie sich nicht erinnern können, ob die Hardware-Uhr auf GMT eingestellt ist, rufen Sie **`hwclock --localtime --show`** auf. Dieses Kommando zeigt die Zeit der Hardware-Uhr an. Wenn sie mit der Zeit auf Ihrer Armbanduhr übereinstimmt, dann ist die Hardware-Uhr auf lokale Zeit eingestellt. Wenn die Zeit der Hardware-Uhr abweicht, ist sie wahrscheinlich auf GMT eingestellt. Sie können das überprüfen, indem Sie die entsprechende anzahl Stunden von der Ausgabe von **hwclock** abziehen bzw. addieren. Wenn Sie zum Beispiel in der Zeitzone MST leben, auch bekannt als GMT-0700, dann addieren Sie sieben Stunden zu der Uhrzeit auf Ihrer Armbanduhr hinzu. Falls es bei Ihnen Sommerzeit gibt, ziehen Sie in den Sommermonaten wieder eine Stunde ab.

Ändern Sie den Wert von `UTC` zu *`0`* (Null), wenn Ihre Hardware-Uhr auf lokale Zeit eingestellt ist.

Legen Sie die neue Datei `/etc/sysconfig/clock` mit dem folgenden Kommando an:

```
cat > /etc/sysconfig/clock << "EOF"
# Begin /etc/sysconfig/clock

UTC=1

# End /etc/sysconfig/clock
EOF
```

Vielleicht möchten Sie sich nun die sehr gute Anleitung unter *http://www.linuxfromscratch.org/hints/downloads/files/time.txt* ansehen. Hier wird erklärt, wie man unter LFS mit der Systemzeit, Zeitzonen, UTC und der Variable `TZ` umgeht.

7.6. Einrichten der Linux Konsole

Dieser Abschnitt behandelt das Bootskript **console**, mit dem die Tastaturbelegung und die Konsoleschriftart eingerichtet wird. Falls Sie nur ASCII-Zeichen verwenden (das Copyright-Symbol, Britische Pfund oder das Euro-Zeichen sind Beispiele für *nicht*-ASCII-Zeichen) und Ihre Tastatur eine US-Amerikanische ist, dann überspringen Sie diesen Abschnitt. Ohne die Konfigurationsdatei unternimmt dieses Bootskript einfach nichts.

Das Skript **console** benutzt `/etc/sysconfig/console` als Konfigurationsdatei. Entscheiden Sie, welche Tastaturbelegung und Bildschirmschriftarten Sie benutzen möchten. Die verschiedenen sprachbezogenen Hilfsdokumente unter *http://www.tldp.org/HOWTO/HOWTO-INDEX/other-lang.html* können Sie bei der Entscheidung unterstützen. Wenn Sie unsicher sind, schauen Sie in `/usr/share/kbd` nach gültigen Tastaturbelegungen und Bildschirmschriften. Lesen Sie die Hilfeseiten `loadkeys(1)` und `setfont(8)` und bestimmen Sie die korrekten Parameter zu diesen Programmen.

Die Datei `/etc/sysconfig/console` sollte Zeilen in der Form: VARIABLE="Wert" enthalten. Die folgenden Variablen sind möglich:

KEYMAP
: 2

KEYMAP_CORRECTIONS
: Diese (wenig eingesetzte) Variable gibt die Argumente für den zweiten Aufruf von **loadkeys** an. Sie ist nützlich, wenn die ausgelieferte Tastaturlayouttabelle nicht zufriedenstellend ist und kleinere Änderungen daran vorgenommen werden sollen. Um z. B. das Euro-Zeichen zu unterstützen, obwohl es normalerweise im Tastaturlayout nicht vorgesehen ist, benutzen Sie den Wert „euro2“.

FONT
: Diese Variable übernimmt die Argumente für das Programm **setfont**. Dies sind üblicherweise der Name der Schrift, „-m“ und der Name der zu ladenden Kodierung. Um beispielsweise die Schrift „lat1-16“ zusammen mit der Kodierung „8859-1“ zu laden, setzen Sie diese Variable auf „lat1-16 -m 8859-1“. Wenn die Variable nicht gesetzt ist, wird das Bootskript **setfont** nicht ausführen und die voreingestellte VGA-Schrift mit der voreingestellten Kodierung wird geladen.

UNICODE
: Setzen Sie diese Variable auf „1“, „yes“ oder „true“, um für die Konsole den UTF-8-Modus zu aktivieren. Dies ist nur für auf UTF-8 basierende Locales sinnvoll und in allen anderen Locales schädlich!

LEGACY_CHARSET
: Für viele Tastaturlayouts gibt es in Kbd keine vorgefertigten Layouttabellen im Unicode-Format. Das Bootskript **console** wird eine verfügbare Tastaturlayouttabelle automatisch in UTF-8 umwandeln, wenn die Variable den Namen der verfügbaren Nicht-UTF-8-Tabelle enthält. Beachten Sie aber bitte, das weder "Tote Tasten" (z. B. Accent + Buchstabe), noch Zusammengesetzte Tasten (Strg + A E) im UTF-8-Modus funktionieren werden, wenn nicht der spezielle Kernel-Patch eingespielt wird. Diese Variable ist nur im UTF-8-Modus nützlich.

BROKEN_COMPOSE
: Setzen Sie diese Variable auf „0“, wenn Sie den Kernel-Patch aus Kapitel 8 installieren möchten. Außerdem müssen Sie den Zeichensatz angeben, den die Compose-Regeln Ihrer Tastaturlayouttabelle erwarten. Diesen Geben Sie in der FONT-Variable hinter dem Parameter „-m“ an. Diese Variable ist nur im UTF-8-Modus nützlich.

Die Unterstützung für direkt in den Kernel eingebaute Tastaturlayouttabellen wurde entfernt, weil es

Berichte über fehlerhafte Ergebnisse gibt.

Einige Beispiele:

- Für eine Nicht-Unicode-Umgebung werden üblicherweise nur die Variablen KEYMAP und FONT benötigt. In Polen würde man dies verwenden:

```
cat > /etc/sysconfig/console << "EOF"
# Begin /etc/sysconfig/console

KEYMAP="pl2"
FONT="lat2a-16 -m 8859-2"

# End /etc/sysconfig/console
EOF
```

- Wie bereits erwähnt, muss das vorbereitete Tastaturlayout manchmal ein wenig angepasst werden. Im folgenden Beispiel wird das Euro-Zeichen zum deutschen Tastaturlayout hinzugefügt:

```
cat > /etc/sysconfig/console << "EOF"
# Begin /etc/sysconfig/console

KEYMAP="de-latin1"
KEYMAP_CORRECTIONS="euro2"
FONT="lat0-16 -m 8859-15"

# End /etc/sysconfig/console
EOF
```

- Und nun folgt ein Beispiel für eine Unicode-Umgebung in Bulgarien, wofür es ein vorbereitetes UTF-8-Tastaturlayout gibt und keine Regeln für Tote bzw. Compose-Tasten:

```
cat > /etc/sysconfig/console << "EOF"
# Begin /etc/sysconfig/console

UNICODE="1"
KEYMAP="bg_bds-utf8"
FONT="LatArCyrHeb-16"

# End /etc/sysconfig/console
EOF
```

- Im vorherigen Beispiel wird die Schrift LatArCyrHeb-16 mit 512 Symbolen eingesetzt. Dies hat zur Folge, das in der Linux-Konsole keine hellen Farben mehr angezeigt werden können, es sei denn man verwendet einen Framebuffer. Falls Sie helle Farben ohne Framebuffer benötigen sollten und dafür mit einigen fehlenden Zeichen leben können (die nicht zur eigenen Sprache gehören), dann können Sie wie folgt auf eine Schrift mit 256 Zeichen zurückgreifen:

```
cat > /etc/sysconfig/console << "EOF"
# Begin /etc/sysconfig/console

UNICODE="1"
KEYMAP="bg_bds-utf8"
FONT="cyr-sun16"

# End /etc/sysconfig/console
EOF
```

- Das nun folgende Beispiel demonstriert die Verwendung der automatischen Umwandlung eines Tastaturlayouts von ISO-8859-15 nach UTF-8 und die Aktivierung von toten Tasten im Unicode-Modus:

```
cat > /etc/sysconfig/console << "EOF"
# Begin /etc/sysconfig/console

UNICODE="1"
KEYMAP="de-latin1"
KEYMAP_CORRECTIONS="euro2"
LEGACY_CHARSET="iso-8859-15"
BROKEN_COMPOSE="0"
FONT="LatArCyrHeb-16 -m 8859-15"

# End /etc/sysconfig/console
EOF
```

- Die nötigen Zeichen für die Sprachen Chinesisch, Japanisch, Koreanisch und ein paar weitere lassen sich in der Linux-Konsole nicht anzeigen. Falls Sie diese benötigen, müssen Sie das X-Window-System, die benötigten Schriften und eine Eingabe-Methode (wie SCIM) installieren.



Anmerkung

Mit der Datei `/etc/sysconfig/console` können Sie ausschließlich die Lokalisierung für die Linux-Textkonsole einrichten. Dies hat nichts mit den Einstellungen für das X-Window-System, SSH-Sitzungen oder einer seriellen Konsole zu tun.

7.7. Einrichten des sysklogd-Skripts

Das `sysklogd`-Skript ruft **syslogd** mit dem Parameter *`-m 0`* auf. Dieser Parameter schaltet die periodische Zeitmarke ab, die sonst von **syslogd** alle 20 Minuten in die Protokolldateien geschrieben wird. Falls Sie diese Zeitmarke wieder einschalten möchten, bearbeiten Sie bitte das Skript `sysklogd` und ändern die Option entsprechend. Für weitere Informationen schlagen Sie bitte in **`man syslogd`** nach.

7.8. Erstellen der Datei /etc/inputrc

Die Datei `inputrc` kümmert sich um das Tastaturmapping in bestimmten Situationen. Sie ist die Konfigurationsdatei von Readline — der Bibliothek, die Eingabe-Funktionen für Bash und die meisten anderen Shells zur Verfügung stellt.

Normalerweise braucht man keine benutzerspezifischen Tastaturmappings, daher erzeugt das folgende Kommando nur die globale Konfigurationsdatei `/etc/inputrc`. Sie wird von jedem Benutzer bzw. der Shell bei der Anmeldung eingelesen und verwendet. Falls Sie später doch eine benutzerspezifische Konfiguration benötigen, können Sie einfach eine Datei mit dem Namen `.inputrc` im Persönlichen Ordner des Benutzers erstellen und dort die angepassten Einstellungen eintragen.

Weitere Informationen zum Anpassen von `inputrc` erhalten Sie mit **info bash** im Abschnitt *Readline Init File*. Eine weitere gute Informationsquelle ist **info readline**.

Sie sehen hier eine generische globale Version der Datei `inputrc`. Darin finden Sie auch erklärende Kommentare zu den verschiedenen Optionen. Beachten Sie bitte, dass sich Kommentare nicht in der gleichen Zeile wie Kommandos befinden dürfen. Erstellen Sie die Datei nun mit dem folgenden Befehl:

```
cat > /etc/inputrc << "EOF"
# Begin /etc/inputrc
# Modified by Chris Lynn <roryo@roryo.dynup.net>

# Allow the command prompt to wrap to the next line
set horizontal-scroll-mode Off

# Enable 8bit input
set meta-flag On
set input-meta On

# Turns off 8th bit stripping
set convert-meta Off

# Keep the 8th bit for display
set output-meta On

# none, visible or audible
set bell-style none

# All of the following map the escape sequence of the
# value contained inside the 1st argument to the
# readline specific functions

"\eOd": backward-word
"\eOc": forward-word

# for linux console
"\e[1~": beginning-of-line
"\e[4~": end-of-line
"\e[5~": beginning-of-history
"\e[6~": end-of-history
"\e[3~": delete-char
"\e[2~": quoted-insert

# for xterm
"\eOH": beginning-of-line
"\eOF": end-of-line
```

```
# for Konsole
"\e[H": beginning-of-line
"\e[F": end-of-line

# End /etc/inputrc
EOF
```

7.9. Die Startdateien von Bash

Das Shell-Programm **/bin/bash** (im weiteren Verlauf nur „shell“ oder „bash“ genannt) benutzt einige Startdateien zum Einrichten der Benutzerumgebung. Jede Datei hat einen bestimmten Zweck und beeinflusst Login- und Interaktiv-Umgebungen unterschiedlich. Die Bash-Dateien in `/etc` enthalten globale Einstellungen. Wenn eine entsprechende Konfigurations-Datei auch im Persönlichen Ordner des Benutzers existiert, überschreibt sie die globalen Einstellungen.

Nach einem erfolgreichen Login wird mit **/bin/login** eine interaktive Login-Shell gestartet. Dazu wird die Datei `/etc/passwd` eingelesen. Eine interaktive nicht-Login-Shell wird von der Kommandozeile aus gestartet (z. B. `[prompt]$`**/bin/bash**). Eine nicht-interaktive Shell findet man üblicherweise bei laufenden Shell-Skripten. Sie ist nicht interaktiv, weil Sie ein Skript abarbeitet und zwischen den Kommandos nicht auf Eingaben vom Benutzer wartet.

Weitere Informationen finden Sie mit **info bash** im Abschnitt *Bash Startup Files and Interactive Shells*.

Die Dateien `/etc/profile` und `~/.bash_profile` werden gelesen, wenn die Shell als interaktive Login-Shell aufgerufen wurde.

Die untenstehende Basisversion der Datei `/etc/profile` stellt ein paar notwendige Umgebungsvariablen für NLS-Unterstützung ein. Eine korrekte Einstellung dieser Variablen bewirkt:

- Die Ausgaben von Programmen werden in die Sprache des Anwenders übersetzt
- Korrekte Einordnung von Zeichen als Buchstaben, Zahlen und weiterer Klassen. Die **bash** benötigt diese Einstellungen, um Sonderzeichen in Befehlszeilen in nicht-englischen Locales verarbeiten zu können.
- Korrekte landesspezifische alphabetische Sortierung
- Passende Papiergröße
- Korrekte Formatierung von Währungs-, Zeit- und Datumswerten

Diese Datei setzt auch die Variable `INPUTRC`. Wenn diese Variable gesetzt ist, benutzen Bash und Readline die vorhin erzeugte Datei `/etc/inputrc`.

Ersetzen Sie `<ll>` mit dem zweistelligen Ländercode für die gewünschte Sprache (z. B. „de“) und `<CC>` mit dem zweistelligen Code für das gewünschte Land (z. B. „DE“ oder „AT“). `<charmap>` sollte durch den korrekten Zeichensatz ersetzt werden, z. B. „iso8859-15“. Auch (optionale) Parameter wie „@euro“ können angehängt werden.

Mit dem folgenden Kommando erhalten Sie eine Liste aller von Glibc unterstützten Locales:

```
locale -a
```

Locales haben häufig mehrere Synonyme. Beispielsweise wird „ISO-8859-1“ häufig auch als „iso8859-1“ und „iso88591“ geschrieben. Einige Programme können nicht mit den verschiedenen Synonymen umgehen, daher ist es das sicherste, den korrekten Namen für ein Locale anzugeben. Um den kanonischen Namen für ein Locale herauszufinden, führen Sie das folgende Programm aus, wobei `<locale name>` die Ausgabe von **locale -a** für Ihr bevorzugtes Locale ist (in diesem Beispiel „de_DE.iso88591“).

```
LC_ALL=<locale-Name> locale-Zeichensatz
```

Für das Locale „de_DE.iso88591“ ergibt das obige Kommando:

```
ISO-8859-1
```

Das endgültige Ergebnis lautet also „de_DE.ISO-8859-1“. Bevor Sie diese Locale-Einstellung allerdings in eine der Startdateien der Bash eintragen, sollten Sie sie testen:

```
LC_ALL=<locale name> locale language
LC_ALL=<locale name> locale charmap
LC_ALL=<locale name> locale int_curr_symbol
LC_ALL=<locale name> locale int_prefix
```

Das obige Kommando sollte Ihnen folgende Daten ausgeben: Land und Sprache, den vom Locale benutzten Zeichensatz, die Währung und den internationalen Telefonnumern-Prefix. Falls eines der Kommandos eine Fehlermeldung wie die folgende ausgibt, dann wurde entweder die Locale in Kapitel 6 nicht installiert, oder wird von der Standardinstallation von Glibc nicht unsterstützt.

```
locale: Cannot set LC_* to default locale: No such file or directory
```

Falls Sie diese oder eine ähnliche Fehlermeldung erhalten, sollten Sie die gewünschte Locale installieren oder eine andere Locale verwenden. Zur Installation der fehlenden Locale benutzen Sie das Programm **localedef**. Alle weiteren Schritte im Buch gehen davon aus, dass Sie keine solche Fehlermeldung wie oben erhalten haben, bzw. dass der Fehler beseitigt wurde.

Es gibt einige Pakete außerhalb von LFS, die Ihre Locale möglicherweise nicht richtig unterstützen. Ein Beispiel dafür ist die X-Bibliothek (Teil des X Window System), die die folgende Meldung ausgibt, wenn der Name für das Locale nicht exakt auf eine der internen Zeichensatztabellen passt:

```
Warning: locale not supported by Xlib, locale set to C
```

In vielen Fällen erwartet Xlib, dass der Name für den Zeichensatz in Großbuchstaben und mit Bindestrichen geschrieben wird. Also "ISO-8859-1" statt "iso88591". Manchmal hilft es auch, den Zeichensatz aus dem Namen der Locale wegzulassen. Dies können Sie mit dem Kommando **locale charmap** in beiden Locales prüfen. Sie würden also "de_DE.ISO-8859-15@euro" durch "de_DE@euro" ersetzen, damit Xlib Ihre Locale versteht.

Möglicherweise haben noch weitere Programme Schwierigkeiten mir Ihrer Locale (und geben vielleicht noch nicht einmal eine Fehlermeldung aus), falls der Name der Locale nicht den Annahmen des Programmierers entspricht. In solchen Fällen kann man versuchen herauszufinden, wie andere Linux-Distributionen mit dem Problem umgehen.

Wenn Sie die korrekten Locale-Einstellungen herausgefunden haben, erstellen Sie die Datei `/etc/profile`:

```
cat > /etc/profile << "EOF"
# Begin /etc/profile

export LANG=<ll>_<CC>.<charmap><@modifiers>
export INPUTRC=/etc/inputrc

# End /etc/profile
EOF
```

Die Locale „C“ (Standard) und „en_US“ (empfohlene Locale für englische Benutzer in den USA) unterscheiden sich. „C“ verwendet den Zeichensatz US-ASCII mit 7 Bit und behandelt Zeichen mit gesetztem hohen Bit als ungültig. Das ist auch der Grund dafür, dass z. B. **ls** diese Zeichen mit einem Fragezeichen darstellt. Auch der Versuch, eine E-Mail mit solchen Zeichen mit Mutt oder Pine zu versenden ergibt eine nicht RFC-konforme Mail (der Zeichensatz in einer solchen Mail ist dann „unknown 8-bit“). Sie können die Locale „C“ also nur einsetzen, wenn Sie sicher sind, dass Sie niemals 8-Bit-Zeichen benötigen.

UTF-8-basierte Locales werden leider von vielen Programmen nicht richtig unterstützt. Das Programm **watch** zeigt in UTF-8-Locales nur ASCII-Zeichen an; diese Beschränkung besteht nicht in normalen 8-Bit-Locales wie en_US. Es wird allerdings daran gearbeitet, solche Probleme zu dokumentieren und zu beheben. Siehe auch: *http://www.linuxfromscratch.org/blfs/view/svn/introduction/locale-issues.html*.

7.10. Einrichten des localnet-Skripts

Eine Teilaufgabe des **localnet**-Skripts ist das Einstellen des Hostnamens. Dieser muss in der Datei `/etc/sysconfig/network` festgelegt werden.

Erstellen Sie die Datei `/etc/sysconfig/network` und geben Sie den Hostnamen ein:

```
echo "HOSTNAME=<lfs>" > /etc/sysconfig/network
```

`<lfs>` muss hier durch den Namen für Ihren Computer ersetzt werden. Geben Sie hier nicht den FQDN (Fully Qualified Domain Name -> Vollständigen Domänennamen) ein. Diesen werden Sie erst später in der Datei `/etc/hosts` eintragen. An dieser Stelle wird nur ein einfacher Rechnername benötigt.

7.11. Anpassen der Datei /etc/hosts

Wenn Sie eine Netzwerkkarte einrichten möchten, müssen Sie eine IP-Adresse, den voll qualifizierten Domänennamen und mögliche Aliasnamen in `/etc/hosts` eintragen. Die Syntax lautet:

```
IP-Adresse meinhost.meinedomain.org aliasname
```

Solange Ihr Computer nicht offiziell im Internet bekannt ist (d. h. Sie haben eine registrierte Domain und einen gültigen zugewiesenen IP-Block, die meisten haben dies nicht), sollten Sie sicherstellen, dass die IP-Adresse im privaten Adressraum liegt. Gültige Adressräume dafür sind:

```
Privater Adressbereich             Normaler Prefix
10.0.0.1 - 10.255.255.254          8
172.x.0.1 - 172.x.255.254          16
192.168.y.1 - 192.168.y.254        24
```

x kann eine Zahl zwischen 16-31 sein. y kann zwischen 0-255 liegen.

Eine gültige private IP-Adresse wäre 192.168.1.1. Ein vollqualifizierter Domänenname wäre beispielsweise lfs.beispiel.de

Selbst wenn Sie keine Netzwerkkarte einrichten müssen Sie einen voll qualifizierten Domänennamen eintragen. Er wird zur korrekten Funktion vieler Programme benötigt.

Erzeugen Sie `/etc/hosts` mit dem folgenden Kommando:

```
cat > /etc/hosts << "EOF"
# Begin /etc/hosts (network card version)

127.0.0.1 localhost
<192.168.1.1> <HOSTNAME.beispiel.de> [alias1] [alias2 ...]

# End /etc/hosts (network card version)
EOF
```

Natürlich müssen Sie *<192.168.1.1>* und *<HOSTNAME.beispiel.de>* nach Ihrem Belieben ändern (bzw. die IP-Adresse und Hostnamen eintragen, die Sie von Ihrem Netzwerkadministrator bekommen haben, falls Ihr Rechner an ein bestehendes Netzwerk angeschlossen wird). Die optionalen Aliasnamen können weggelassen werden.

Wenn Sie keine Netzwerkkarte einrichten, erzeugen Sie `/etc/hosts` mit diesem Kommando:

```
cat > /etc/hosts << "EOF"
# Begin /etc/hosts (no network card version)

127.0.0.1 <HOSTNAME.beispiel.de> <HOSTNAME> localhost

# End /etc/hosts (no network card version)
EOF
```

7.12. Erzeugen von benutzerdefinierten symbolischen Links zu Geräten

7.12.1. Symbolische Links für CD-ROMs

Einige von den Programmen, die Sie vielleicht später installieren möchten, erwarten die Existenz von /dev/cdrom und /dev/dvd (z. B. einige Media-Player). Außerdem könnte es praktischer sein, diese symbolischen Links in `/etc/fstab` zu verwenden. Suchen Sie für jedes CD-ROM-Laufwerk den zugehörigen Ordner in `/sys` (dies könnte `/sys/block/hdd` sein) und führen Sie ein solches Kommando aus:

```
udevtest /block/hdd
```

Halten Sie Ausschau nach den Zeilen mit den verschiedenen *_id-Programmen.

Es gibt zwei Methoden zum Erzeugen symbolischer Links. Die erste Möglichkeit basiert auf dem Modell und der Seriennummer des Geräts, die zweite basiert auf dem Anschluss und dem Bus des Geräts. Wenn Sie die erste Variante einsetzen möchten, erstellen Sie eine Datei wie folgt:

```
cat >/etc/udev/rules.d/82-cdrom.rules << EOF

# Custom CD-ROM symlinks
SUBSYSTEM=="block", ENV{ID_MODEL}=="SAMSUNG_CD-ROM_SC-148F", \
    ENV{ID_REVISION}=="PS05", SYMLINK+="cdrom"
SUBSYSTEM=="block", ENV{ID_MODEL}=="PHILIPS_CDD5301", \
    ENV{ID_SERIAL}=="5VO1306DM00190", SYMLINK+="cdrom1 dvd"

EOF
```



Anmerkung

Die Beispiele in diesem Buch funktionieren korrekt. Beachten Sie aber bitte, dass udev den linksgerichteten Schrägstrich nicht als Zeilenfortsetzung interpretiert. Wenn Sie udev-Regeln anpassen, dann muss jede Regel in einer einzigen physikalischen Zeile stehen.

Auf diese Weise bleib der symbolische Link stabil, selbst wenn Sie das Gerät an einem anderen Stecker am IDE-Kabel anschließen. Der Link `/dev/cdrom` wird allerdings nicht mehr erzeugt, wenn Sie das SAMSUNG-Laufwerk durch ein anderes ersetzen.

Der Schlüssel SUBSYSTEM=="block" ist nötig, um das Erstellen unnötiger generischer SCSI-Geräte zu vermeiden. Ohne ihn würden bei SCSI-CD-ROMs manchmal symbolische Links zu dem richtigen Gerät `/dev/srX`, aber manchmal auch zu `/dev/sgX` erzeugt werden (was falsch wäre).

Die zweite Methode sieht so aus:

```
cat >/etc/udev/rules.d/82-cdrom.rules << EOF

# Custom CD-ROM symlinks
SUBSYSTEM=="block", ENV{ID_TYPE}=="cd", \
    ENV{ID_PATH}=="pci-0000:00:07.1-ide-0:1", SYMLINK+="cdrom"
SUBSYSTEM=="block", ENV{ID_TYPE}=="cd", \
    ENV{ID_PATH}=="pci-0000:00:07.1-ide-1:1", SYMLINK+="cdrom1 dvd"

EOF
```

Auf diese Weise bleibt der Link stabil, wenn Sie das Laufwerk durch ein anderes ersetzen. Dies funktioniert aber nicht, wenn Sie das Laufwerk an einer anderen Stelle im Bus anschließen. Der Schlüssel ENV{ID_TYPE}=="cd" stellt sicher, dass der symbolische Link verschwindet, wenn Sie etwas anderes als ein CD-ROM an dieser Position im Bus anschließen.

Natürlich können Sie diese beiden Möglichkeiten auch kombinieren.

7.12.2. Der Umgang mit doppelten Geräten

In Abschnitt 7.4, „Umgang mit Geräten und Modulen an einem LFS-System“ wurde ja bereits erwähnt, dass die Reihenfolge, in der Geräte in `/dev` angelegt werden, vollkommen zufällig sein kann. Nehmen wir an Sie haben eine USB-Webcam und eine USB-TV-Tuner, so zeigt `/dev/video0` auf die Kamera und `/dev/video1` auf den Tuner. Manchmal kann sich die Reihenfolge bei einem Neustart aber auch einfach umkehren. Dieses Phänomen kann man für alle Geräte außer Sound- und Netzwerkkarten mittels Udev-Regeln und symbolischen Links lösen. Wie man dies mit Netzwerkkarten löst, steht in Abschnitt 7.13, „Einrichten des network-Skripts“ beschrieben, und die Anleitung für Soundkarten finden Sie in *BLFS*.

Sie sollten für jedes der möglicherweise problematischen Geräte (selbst wenn das Problem mit Ihrer bisherigen Linux-Distribution nicht auftritt) den passenden Ordner unter `/sys/class` oder `/sys/block` suchen. Videogeräte finden Sie unter `/sys/class/video4linux/videoX`. Finden Sie die Attribute, die das Gerät unverwechselbar erkennbar machen (üblicherweise Hersteller- und Produkt-IDs und/oder Seriennummern):

```
udevinfo -a -p /sys/class/video4linux/video0
```

Schreiben Sie nun die passenden Regel zum Erzeugen der symbolischen Links:

```
cat >/etc/udev/rules.d/83-duplicate_devs.rules << EOF

# Persistent symlinks for webcam and tuner
KERNEL=="video*", SYSFS{idProduct}=="1910", SYSFS{idVendor}=="0d81", \
    SYMLINK+="webcam"
KERNEL=="video*", SYSFS{device}=="0x036f", SYSFS{vendor}=="0x109e", \
    SYMLINK+="tvtuner"

EOF
```

Als Ergebnis erhalten Sie immer noch die Gerätedateien `/dev/video0` und `/dev/video1`, die jeweils unterschiedliche Geräte meinen können (und deshalb nicht direkt angesprochen werden sollten). Zusätzlich erhalten Sie aber auch die symbolischen Links `/dev/tvtuner` und `/dev/webcam`, und diese zeigen immer auf das richtige Gerät.

Weitere Informationen zum Schreiben von Udev-Regeln finden Sie in `/usr/share/doc/udev-096/index.html`.

7.13. Einrichten des network-Skripts

Diesen Abschnitt müssen Sie nur lesen, wenn Sie eine Netzwerkkarte einrichten möchen.

Wenn Sie keine Netzwerkkarte haben, brauchen Sie höchstwahrscheinlich keine Konfigurationsdateien bezüglich Netzwerkkarten einrichten. In diesem Fall sollten Sie alle symbolischen Links mit Namen `network` aus den Runlevel-Ordnern entfernen (`/etc/rc.d/rc*.d`).

7.13.1. Einrichten von stabilen Namen für Netzwerkkarten

Die hier aufgeführten Anweisungen sind natürlich optional, wenn Sie nur eine Netzwerkkarte haben.

Mit Udev und modularen Netzwerktreibern ist keine stabile Durchnummerierung von Netzwerkkarten über Neustarts hinweg gewährleistet. Dies liegt daran, dass die Treiber parallel geladen werden und die Reihenfolge daher unvorhersagbar ist. Wenn ein Rechner z. B. eine Netzwerkkarte von Intel und eine von Realtek hat, so könne die Intel-Karte `eth0` und die Realtek-Karte `eth1` heißen. In manchen Fällen könnten die Karten nach einem Neustart aber genau umgekehrt zugewiesen worden sein. Um dieses Problem zu umgehen, sollten Sie Udev-Regeln erstellen und stabile Namen zu den Netzwerkkarten basierend auf deren MAC-Adresse oder Bus-Position zuweisen.

Wenn Sie die MAC-Adresse zur Identifikation der Netzwerkkarten benutzen möchten, dann können Sie diese mit dem folgenden Kommando herausfinden:

```
grep -H . /sys/class/net/*/address
```

Geben Sie jeder Netzwerkkarte (außer dem Loopback-Gerät) einen beschreibenden Namen (Beispiel: „realtek“) und erzeugen Udev-Regeln wie folgt:

```
cat > /etc/udev/rules.d/26-network.rules << EOF
ACTION=="add", SUBSYSTEM=="net", SYSFS{address}=="00:e0:4c:12:34:56", \
    NAME="realtek"
ACTION=="add", SUBSYSTEM=="net", SYSFS{address}=="00:a0:c9:78:9a:bc", \
    NAME="intel"
EOF
```



Anmerkung

Die Beispiele in diesem Buch funktionieren korrekt. Beachten Sie aber bitte, dass udev den linksgerichteten Schrägstrich nicht als Zeilenfortsetzung interpretiert. Wenn Sie udev-Regeln anpassen, dann muss jede Regel in einer einzigen physikalischen Zeile stehen.

Wenn Sie die Bus-Position als Schlüssel verwenden möchten, erstellen Sie eine Udev-Regel wie folgt:

```
cat > /etc/udev/rules.d/26-network.rules << EOF
ACTION=="add", SUBSYSTEM=="net", BUS=="pci", ID=="0000:00:0c.0", \
    NAME="realtek"
ACTION=="add", SUBSYSTEM=="net", BUS=="pci", ID=="0000:00:0d.0", \
    NAME="intel"
EOF
```

Diese Regeln werden die Netzwerkkarten immer zu „realtek“ und „intel“ umbenennen, unabhängig von der ursprünglichen Nummerierung des Kernels. Die ursprünglichen Gerätenamen „eth0“ und „eth1“ existieren also nicht mehr, es sei denn Sie vergeben einen solchen „beschreibenden“ Namen im Schlüsselwort NAME ein. Benutzen Sie zur Einrichtung des Netzwerks nun also die beschreibenden Namen aus den Udev-Regeln anstelle von „eth0“.

Beachten Sie, dass die obigen Regeln nicht in jeder Systemkonfiguration funktionieren. MAC-basierte Regeln funktionieren z. B. nicht mit Bridges oder VLANs, weil diese die gleiche MAC-Adresse wie die zugehörige physikalische Netzwerkkarte verwenden. Wenn Sie virtuelle Netzwerkschnittstellen einsetzen, haben Sie zwei Möglichkeiten. Die eine ist der Schlüssel DRIVER="?*" nach SUBSYSTEM=="net" in MAC-basierten Regeln. Dies funktioniert nicht mit einigen älteren Netzwerkkarten, weil die Treiber keine DRIVER-Variable im uevent exportieren. Eine weitere Möglichkeit ist es, Bus-basierte Regeln zu verwenden.

Der zweite nicht funktionierende Fall sind Drahtlos-Karten die den MadWifi- oder HostAP-Treiber verwenden, weil diese zumindest zwei Schnittstellen mit der gleichen MAC-Adresse an der gleichen Bus-Position erzeugen. Der Madwifi-Treiber legt z. B. die Schnittstellen athX und wifiX an (wobei X für eine Zahl steht). Um in diesem Fall eine Unterscheidung machen zu können, verwenden Sie am besten den Schlüssel KERNEL=="ath*" nach SUBSYSTEM=="net".

Es könnte noch weitere Fälle geben, in denen die obigen Regeln nicht richtig funktionieren. Derartige Fehler werden zur Zeit an die Linux-Distributionen weitergeleitet, aber derzeit gibt es noch keine Lösung, die wirklich alle Fälle abdeckt.

7.13.2. Erstellen der Konfigurationsdateien für Netzwerkgeräte

Welche Netzwerkgeräte von den Skripten gestartet und gestoppt werden, hängt von den Dateien und Ordnern in `/etc/sysconfig/network-devices` ab. Dieser Ordner sollte pro Netzwerkgerät einen Unterordner in der Form `ifconfig.xyz` enthalten, wobei „xyz“ der Name des Netzwerkgerätes ist (zum Beispiel eth0 oder eth0:1).

Das folgende Kommando erzeugt die Beispieldate `ipv4` für *eth0*:

```
cd /etc/sysconfig/network-devices &&
mkdir -v ifconfig.eth0 &&
cat > ifconfig.eth0/ipv4 << "EOF"
ONBOOT=yes
SERVICE=ipv4-static
IP=192.168.1.1
GATEWAY=192.168.1.2
PREFIX=24
BROADCAST=192.168.1.255
EOF
```

Natürlich müssen die Werte der Variablen in jeder Datei angepasst werden um mit Ihrer tatsächlichen Systemkonfiguration übereinzustimmen. Wenn die `ONBOOT`-Variable auf „yes“ gesetzt ist, wird das network-Skript die Netzwerkkarte beim booten starten. Wenn sie auf irgendeinen anderen Wert gesetzt wird, ignoriert das Skript dieses Gerät und startet es dementsprechend auch nicht.

Der Eintrag `SERVICE` legt fest, wie die IP-Adresse vergeben wird. Die LFS-Bootskripte sind in Bezug auf IP-Adressen-Zuordnung modular aufgebaut. Durch das Erstellen weiterer Dateien in `/etc/sysconfig/network-devices/services` können Sie weitere Zuweisungsmethoden definieren. Das könnten Sie z. B. tun, um eine IP-Adresse über DHCP zu beziehen (dies wird im BLFS-Buch beschrieben).

Die Variable `GATEWAY` sollte die IP-Adresse Ihres Standard-Gateways enthalten. Wenn Sie kein Standard-Gateway haben, setzen Sie ein Kommentarzeichen vor die Zeile (#).

`PREFIX` muss die Anzahl der verwendeten Bits in der Netzwerkmaske enthalten. Jedes Oktett hat acht Bit. Wenn die Netzwerkmaske 255.255.255.0 lautet, dann werden die ersten drei Oktette benutzt (3x8 = 24 Bit) um das Netzwerk zu bezeichnen. 255.255.255.240 benutzt die ersten 28 Bit. Prefixe mit mehr als 24 Bit werden häufig von DSL- und Kabelbasierten Internet-Dienstleistern (ISP) verwendet. In diesem Beispiel (PREFIX=24) ist die Netzwerkmaske 255.255.255.0. Passen Sie sie Ihrem Subnetz entsprechend an.

7.13.3. Erstellen der Datei /etc/resolv.conf

Wenn Sie mit dem Internet verbunden sind, brauchen Sie höchstwahrscheinlich DNS-Namensauflösung um Internet Domänennamen zu IP-Adressen aufzulösen. Dies erreichen Sie am einfachsten, indem Sie die IP-Adresse des DNS-Servers (stellt Ihr Internet-Provider oder Netzwerkadministrator bereit) in `/etc/resolv.conf` eintragen. Erzeugen Sie die Datei mit diesem Kommando:

```
cat > /etc/resolv.conf << "EOF"
# Begin /etc/resolv.conf

domain {<Ihr Domänenname>}
nameserver <IP-Adresse des primären Nameservers>
nameserver <IP-Adresse des sekundären Nameservers>

# End /etc/resolv.conf
EOF
```

Natürlich müssen Sie *<IP-Adresse des primären Nameservers>* durch die echte IP-Adresse Ihres primären DNS-Servers ersetzen. Oftmals gibt es mehr als einen Eintrag (offizielle Nameserver müssen aus Fallback-Gründen immer auch einen sekundären DNS-Server haben). Die IP-Adresse könnte auch die eines Routers in Ihrem lokalen Netzwerk sein. Wenn Sie keinen zweiten Nameserver haben oder möchten, entfernen Sie den zweiten *nameserver*-Eintrag.

Kapitel 8. Das LFS-System bootfähig machen

8.1. Einführung

Nun ist es an der Zeit Ihr LFS bootfähig zu machen. In diesem Kapitel erstellen Sie die Datei `fstab`, einen neuen Kernel für Ihr LFS-System und Sie installieren den GRUB Bootloader, damit Sie Ihr LFS-System zum booten auswählen können.

8.2. Erstellen der Datei /etc/fstab

Die Datei `/etc/fstab` wird von einigen Programm benutzt, um festzustellen, wo und in welcher Reihenfolge Partitionen eingehängt werden sollen und welche Dateisysteme geprüft werden müssen. Erstellen Sie nun eine neue Tabelle der Dateisysteme:

```
cat > /etc/fstab << "EOF"
# Begin /etc/fstab

# file system  mount-point  type   options         dump  fsck
#                                                        order

/dev/<xxx>     /            <fff>  defaults        1     1
/dev/<yyy>     swap         swap   pri=1           0     0
proc           /proc        proc   defaults        0     0
sysfs          /sys         sysfs  defaults        0     0
devpts         /dev/pts     devpts gid=4,mode=620  0     0
shm            /dev/shm     tmpfs  defaults        0     0
# End /etc/fstab
EOF
```

Natürlich müssen Sie `<xxx>`, `<yyy>` und `<fff>` mit den korrekten Werten für Ihr System ersetzen — zum Beispiel `hda2`, `hda5` und `ext3`. Die Details zu den sechs Feldern in dieser Tabelle finden Sie mittels **man 5 fstab**.

Der Mountpunkt `/dev/shm` für das `tmpfs`-Dateisystem wird hier eingefügt um POSIX-konformes shared memory zu gewährleisten. Ihr Kernel muss Unterstützung dafür haben damit das funktioniert — mehr darüber finden Sie im nächsten Abschnitt. Beachten Sie bitte, dass zur Zeit nur wenige Programme POSIX shared memory verwenden. Daher können Sie den Mountpunkt `/dev/shm` als optional betrachten. Mehr Informationen dazu finden Sie in `Documentation/filesystems/tmpfs.txt` im Quellordner Ihrer Kernel-Quellen.

Dateisysteme die ursprünglich aus MS-DOS oder Windows stammen (das sind: vfat, ntfs, smbfs, cifs, iso9660, udf) müssen mit dem mount-Parameter „iocharset“ eingebunden werden, damit Nicht-Ascii-Zeichen in Dateinamen korrekt gehandhabt werden können. Der Wert des Parameters sollte Ihrer Locale entsprechen, so angepasst, dass der Kernel ihn verstehen kann. Dies funktioniert nur, wenn der nötige Zeichensatz (zu finden unter File systems -> Native Language Support) in den Kernel eingebaut oder als Modul kompiliert ist. Der Parameter „codepage“ ist desweiteren für vfat- und smbfs-Dateisysteme erforderlich. Der Wert sollte der in Ihrem Land unter MS-DOS verwendeten Codepage entsprechen. Um beispielsweise einen USB-Stick in ru_RU.KOI8-R einzubinden, muss der Benutzer diese Zeile in `/etc/fstab` eintragen:

```
/dev/sda1    /media/flash vfat
    noauto,user,quiet,showexec,iocharset=koi8r,codepage=866 0 0
```

Die entsprechende Zeile für ru_RU.UTF-8 lautet:

```
/dev/sda1    /media/flash vfat
    noauto,user,quiet,showexec,iocharset=utf8,codepage=866 0 0
```



Anmerkung

Im letzteren Fall wird der Kernel die folgende Meldung ausgeben:

```
FAT: utf8 is not a recommended IO charset for FAT filesystems,
    filesystem will be case sensitive!
```

Diese Meldung sollte einfach ignoriert werden, da alle anderen Werte für „iocharset“ zu einer fehlerhaften Darstellung der Dateinamen in UTF-8 führen würden.

Es ist ebenso möglich, die Werte für codepage und iocharset für bestimmte Dateisysteme bereits bei der Kernelkonfiguration festzulegen. Die nötigen Parameter finden Sie unter „Default NLS Option“ (CONFIG_NLS_DEFAULT), „Default Remote NLS Option“ (CONFIG_SMB_NLS_DEFAULT), „Default codepage for FAT“ (CONFIG_FAT_DEFAULT_CODEPAGE) und „Default iocharset for FAT“ (CONFIG_FAT_DEFAULT_IOCHARSET). Für das NTFS-Dateisystem gibt es derzeit keine Möglichkeit, die Werte in der Kernelkonfiguration vorzugeben.

8.3. Linux-2.6.16.27

Das Paket Linux enthält den Linux-Kernel.

Geschätzte Kompilierzeit: 1.5 - 3 SBU
Ungefähr benötigter Festplattenplatz: 310 - 350 MB

8.3.1. Installation des Kernel

Kompilieren und Installieren des Kernels sind im Grunde nur ein paar Schritte — Konfigurieren, kompilieren und installieren. Falls Sie die hier benutzte Methode nicht mögen, schauen Sie in der Datei `README` im Kernel-Quellordner nach Alternativen.

Der Standard-Kernel erzeugt fehlerhafte Bytes, wenn im UTF-8-Modus sog. Tote Tasten gedrückt werden. Außerdem kann man im UTF-8-Modus nur ASCII-Zeichen kopieren und einfügen. Mit dem folgenden Patch können Sie diese Probleme beheben:

```
patch -Np1 -i ../linux-2.6.16.27-utf8_input-1.patch
```

Bereiten Sie den Kompiliervorgang mit dem folgenden Kommando vor:

```
make mrproper
```

Hierdurch wird sichergestellt, dass der Kernel-Baum absolut sauber ist. Das Kernel-Team empfiehlt, dieses Kommando vor *jedem* Kompilieren des Kernels auszuführen. Sie sollten sich nicht darauf verlassen, dass die Quellen nach dem Entpacken sauber sind.

Richten Sie den Kernel nun mit der menügeführten Oberfläche ein. In BLFS finden Sie unter *http://www.linuxfromscratch.org/blfs/view/svn/longindex.html#kernel-config-index* einige Informationen zu bestimmten Kernel-Voraussetzungen von Software außerhalb von LFS:

```
make menuconfig
```

`make oldconfig` könnte in einigen Fällen besser geeignet sein. Schauen Sie in die Datei `README` um mehr Informationen zu erhalten.

Wenn Sie möchten, können Sie die Kernelkonfiguration überspringen und einfach die Kernel-Konfigurationsdatei `.config` von Ihrem Host-System nach `linux-2.6.16.27` kopieren (falls sie verfügbar ist). Das wird allerdings nicht empfohlen, Sie sind besser dran, wenn Sie alle Konfigurationsmenüs durchsehen und Ihre eigene Kernelkonfiguration einrichten.

Kompilieren Sie das Kernel-Abbild und die Module:

```
make
```

Wenn Sie Kernel-Module verwenden, brauchen Sie wahrscheinlich die Datei `/etc/modprobe.conf`. Informationen zu Modulen und Kernelkonfiguration im Allgemeinen finden Sie unter Abschnitt 7.4, „Umgang mit Geräten und Modulen an einem LFS-System“ und in der Dokumentation zum Kernel `linux-2.6.16.27`. Auch `modprobe.conf(5)` enthält nützliche Informationen.

Installieren Sie die Module, falls Ihre Kernelkonfiguration solche verwendet:

```
make modules_install
```

Das Kompilieren des Kernel ist nun abgeschlossen, aber einige der erzeugten Dateien befinden sich noch im Quellordner. Um die Installation abzuschließen, müssen Sie noch ein paar Dateien in den Ordner `/boot` kopieren.

Der Pfad zur Kerneldatei variiert, abhängig von der benutzten Plattform auf der Sie arbeiten. Das folgende Kommando geht von einem x86-System aus:

```
cp -v arch/i386/boot/bzImage /boot/lfskernel-2.6.16.27
```

`System.map` ist eine Symboldatei für den Kernel. Sie ordnet Funktions-Einstiegspunkte jeder Funktion in der Kernel-API sowie Adressen der Kernel-Datenstrukturen zu. Geben Sie das folgende Kommando ein, um die Datei zu installieren:

```
cp -v System.map /boot/System.map-2.6.16.27
```

`.config` ist die Kernel-Konfigurationsdatei, die durch das obige Kommando **make menuconfig** erzeugt wurde. Sie enthält alle Konfigurationsoptionen für den soeben kompilierten Kernel. Es ist sinnvoll, diese Datei aufzubewahren:

```
cp -v .config /boot/config-2.6.16.27
```

Installieren Sie die Dokumentation zum Linux-Kernel:

```
install -d /usr/share/doc/linux-2.6.16.27 &&
cp -r Documentation/* /usr/share/doc/linux-2.6.16.27
```

Beachten Sie bitte, dass die Dateien im Kernel-Quellordner nicht `root` gehören. Immer wenn Sie ein Paket als `root`-Benutzer entpacken (so wie Sie es hier im chroot tun), erhalten die entpackten Dateien die Benutzer- und Gruppen ID desjenigen, der das Archiv erstellt hat. Das ist üblicherweise für normale Pakete kein Problem weil Sie den Quellordner nach der Installation löschen. Aber die Linux-Quellen liegen oft sehr lange auf Ihrem Computer, daher ist die Chance groß, dass ein zukünftiger Benutzer auf Ihrem System die Benutzer-ID erhält, die Ihre Kernel-Quellen derzeit haben, und damit wäre er der Besitzer dieser Dateien und hätte dann auch Schreibrechte darauf.

Wenn Sie die Kernelquellen aufbewahren möchten, sollten Sie **chown -R 0:0** auf den Ordner `linux-2.6.16.27` anwenden. So stellen Sie sicher, dass alle Dateien dem Benutzer `root` gehören.



Warnung

Einige Kerneldokumentationen empfehlen das Erzeugen eines Links von `/usr/src/linux` auf den Ordner mit den Kernelquellen. Dies bezieht sich aber nur auf Kernel vor der 2.6er Serie zu und *darf nicht* in einem LFS-System angewendet werden. Es verursacht Probleme beim Kompilieren von Paketen die Sie vielleicht im Nachhinein noch installieren möchten.

Die Header in Ihrem Systemordner `include` sollten *immer* diejenigen sein, mit denen die Glibc kompiliert wurde (also die Linux-Libc-Header) und dürfen daher bei einem Kernelupgrade *keinesfalls* durch die neuen Kernel-Header ersetzt werden.

8.3.2. Inhalt von Linux

Installierte Dateien: config-2.6.16.27, lfskernel-2.6.16.27 und System.map-2.6.16.27

Kurze Beschreibungen

`config-2.6.16.27`	Enthält alle ausgewählten Konfigurationsoptionen für den Kernel.
`lfskernel-2.6.16.27`	Dies ist der Kernel, der Motor Ihres GNU/Linux-Systems. Nach dem Einschalten Ihres Rechners ist der Kernel der erste Teil des Betriebssystems, der geladen wird. Er erkennt und initialisiert alle Komponenten Ihrer Computer-Hardware und macht diese Komponenten für die Software verfügbar. Er verwandelt eine einzelne CPU in eine Multitasking-Maschine die unzählige Programme scheinbar zur gleichen Zeit ausführen kann.
`System.map-2.6.16.27`	Enthält eine Liste von Adressen und Symbolen. Sie ordnet Einstiegspunkte und Adressen aller Funktionen und Datenstrukturen dem entsprechenden Kernel zu.

8.4. Das LFS-System bootfähig machen

Ihr frisches LFS-System ist nun beinahe fertig. Sie müssen nun noch sicherstellen, dass es booten kann. Die untenstehende Anleitung gilt nur für Computer mit IA-32-Architektur, dazu gehören alle handelsüblichen PCs. Informationen zum „boot loading“ auf anderen Architekturen finden Sie in den üblichen Dokumentationsquellen zu diesen Architekturen.

Booten kann ein sehr komplexes Thema sein. Hier erstmal ein paar warnende Worte: Sie sollten mit Ihrem jetzigen Bootloader und den Betriebssystemen, die Sie weiter verwenden wollen, vertraut sein. Halten Sie bitte eine „Notfalldiskette“ bereit, damit Sie Ihren Computer starten können, falls Ihr Computer aus irgendwelchen Gründen unbrauchbar wird (weil er zum Beispiel nicht mehr bootet).

Den Grub Bootloader haben Sie bereits installiert. Jetzt müssen ein paar Grub-Dateien an spezielle Orte auf der Festplatte kopiert werden. Bevor Sie das tun, sollten Sie eine Boot-Diskette mit Grub erstellen, nur für den Fall der Fälle. Legen Sie eine leere Diskette ein und führen Sie dieses Kommando aus:

```
dd if=/boot/grub/stage1 of=/dev/fd0 bs=512 count=1
dd if=/boot/grub/stage2 of=/dev/fd0 bs=512 seek=1
```

Entfernen Sie die Diskette und bewahren Sie sie an einem sicheren Ort auf. Starten Sie nun die **grub**-Shell:

```
grub
```

Grub verwendet zur Benennung von Festplatten und Partitionen ein eigenes Schema der Form *(hdn,m)*, wobei *n* die Nummer der Festplatte, und *m* die Nummer der Partition ist. Beide Werte beginnen bei Null. Das bedeutet, dass zum Beispiel die Partition `hda1` für GRUB *(hd0,0)* ist, und `hdb2` ist *(hd1,1)*. Anders als Linux, betrachtet GRUB CD-Rom Laufwerke nicht als Festplatte. Wenn Sie also ein CD-Rom Laufwerk auf `hdb` haben und eine zweite Festplatte auf `hdc`, dann ist die zweite Festplatte immernoch *(hd1)*.

Bestimmen Sie mit den obigen Informationen den Namen Ihrer root-Partition. Im folgenden Beispiel wird angenommen, dass Ihre root-Partition `hda4` ist.

Sagen Sie GRUB zuerst, wo die `stage{1,2}`-Dateien zu finden sind—Sie können die Tabulator-Taste verwenden damit Grub Alternativen anzeigt:

```
root (hd0,3)
```



Warnung

Das nächste Kommando überschreibt Ihren bisherigen Bootloader. Wenn Sie das nicht wollen, führen Sie das Kommando nicht aus. Zum Beispiel wenn Sie einen Bootloader von einem Dritthersteller benutzen möchten um Ihren MBR (Master Boot Record) zu verwalten. In dem Fall würde es Sinn machen, Grub in den „Bootsektor“ Ihrer LFS-Partition zu installieren, das folgende Kommando würde dann lauten: **setup (hd0,3)**.

Weisen Sie GRUB nun an, sich in den MBR von `hda` zu installieren:

```
setup (hd0)
```

Wenn alles in Ordnung ist, wird GRUB nun berichten, dass die nötigen Dateien in `/boot/grub` gefunden wurden. Das ist alles soweit, beenden Sie die **grub**-Shell:

```
quit
```

Nun müssen Sie eine „Menü-Liste“ erstellen. Sie definiert das Bootmenü von Grub:

```
cat > /boot/grub/menu.lst << "EOF"
# Begin /boot/grub/menu.lst

# By default boot the first menu entry.
default 0

# Allow 30 seconds before booting the default.
timeout 30

# Use prettier colors.
color green/black light-green/black

# The first entry is for LFS.
title LFS 6.2
root (hd0,3)
kernel /boot/lfskernel-2.6.16.27 root=/dev/hda4
EOF
```

Vielleicht möchten Sie einen weiteren Eintrag für Ihr Host-System vornehmen. Dieser könnte z. B. so aussehen:

```
cat >> /boot/grub/menu.lst << "EOF"
title Red Hat
root (hd0,2)
kernel /boot/kernel-2.6.5 root=/dev/hda3
initrd /boot/initrd-2.6.5
EOF
```

Falls Sie Windows dual-booten möchten, könnte der folgende Eintrag hilfreich sein:

```
cat >> /boot/grub/menu.lst << "EOF"
title Windows
rootnoverify (hd0,0)
chainloader +1
EOF
```

Falls Ihnen **info grub** nicht alle benötigten Informationen gibt, finden Sie mehr dazu auf den GRUB-Webseiten unter *http://www.gnu.org/software/grub/*.

FHS setzt voraus, das GRUB's `menu.lst` nach `/etc/grub/menu.lst` verlinkt sein sollte. Um diese Voraussetzung zu erfüllen, führen Sie das folgende Kommando aus:

```
mkdir -v /etc/grub &&
ln -sv /boot/grub/menu.lst /etc/grub
```

Kapitel 9. Ende

9.1. Ende

Herzlichen Glückwunsch! Sie sind fertig mit der Installation Ihres eigenen LFS-Systems. Wir wünschen Ihnen viel Freude mit Ihrem brandneuen selbstgebauten Linux.

Sie sollten nun noch die Datei `/etc/lfs-release` erstellen. Mit ihr ist es für Sie (und für uns, wenn Sie uns bei etwas um Hilfe bitten sollten) einfach, herauszufinden, welche LFS-Version Sie haben. Erstellen Sie die Datei mit diesem Kommando:

```
echo 6.2 > /etc/lfs-release
```

9.2. Lassen Sie sich zählen

Sie haben nun das ganze Buch durchgearbeitet. Vielleicht möchten Sie sich jetzt als LFS-Benutzer zählen lassen?! Besuchen Sie *http://www.linuxfromscratch.org/cgi-bin/lfscounter.cgi* und registrieren Sie sich als LFS-Benutzer indem Sie Ihren Namen und die Versionsnummer Ihres ersten LFS-Systems dort eintragen.

Lassen Sie uns nun Ihr LFS booten...

9.3. Neustarten des Systems

Nachdem nun sämtliche Software installiert ist, wird es Zeit, den Computer neu zu starten. Sie sollten allerdings ein paar Dinge beachten. Das bisher erstellte System ist absolut minimal und hat höchstwahrscheinlich nicht genügend Funktionen, um ernsthaft damit arbeiten zu können. Während Sie weiterhin in der chroot-Umgebung sind, können Sie Pakete aus dem BLFS-Buch installieren. Das versetzt Sie in eine weitaus bessere Lage nach dem Neustart Ihres Systems. Wenn Sie einen textbasierten Webbrowser wie z. B. Lynx installieren, können Sie das BLFS-Buch in einer virtuellen Konsole lesen und in einer anderen Pakete kompilieren. Mit GPM können Sie auch Kopieren und Einfügen zwischen den Konsolen nutzen. Zusätzlich können Sie auch Pakete wie Dhcpd oder PPP installieren. Dies ist z. B. dann nützlich, wenn Sie keine statische IP-Adresse nutzen können.

Nachdem dies gesagt ist, können Sie nun in Ihr frisch installiertes System booten. Als erstes verlassen Sie die chroot-Umgebung:

```
logout
```

Hängen Sie die virtuellen Dateisysteme aus:

```
umount -v $LFS/dev/pts
umount -v $LFS/dev/shm
umount -v $LFS/dev
umount -v $LFS/proc
umount -v $LFS/sys
```

Und hängen Sie das LFS-Dateisystem aus:

```
umount -v $LFS
```

Falls Sie sich zu Beginn für mehrere Partitionen entschieden haben, müssen Sie die anderen Partitionen aushängen, bevor Sie die Hauptpartition aushängen:

```
umount -v $LFS/usr
umount -v $LFS/home
umount -v $LFS
```

Jetzt können Sie Ihren Computer neu starten:

```
shutdown -r now
```

Unter der Annahme, dass der GRUB Bootloader wie vorgeschlagen installiert wurde, sollte das Standard-Bootmenü automatisch *LFS 6.2* booten.

Nach dem Neustart ist Ihr LFS-System bereit; Sie können es nun benutzen und mit der Installation weiterer Software beginnen.

9.4. Was nun?

Vielen Dank, dass Sie dieses Buch gelesen haben. Wir hoffen, dass Sie es nützlich fanden und viel über die Installation von Linux gelernt haben.

Nachdem Sie nun mit der Installation von LFS fertig sind, fragen Sie sich vielleicht: „Was kommt nun?“. Um diese Frage zu beantworten haben wir eine Reihe von Links für Sie zusammengestellt.

- Pflege und Wartung

  Für jede Software werden regelmäßig Sicherheitslücken und Fehler gemeldet. Da ein LFS aus den Quellen kompiliert ist, liegt es an Ihnen, diese Berichte zu verfolgen. Es gibt dazu verschiedene Online-Ressourcen die Sie sich ansehen können:

  - Freshmeat.net (*http://freshmeat.net/*)

    Freshmeat kann Sie (vie E-Mail) über neue Programmversionen informieren.

  - *CERT* (Computer Emergency Response Team)

    CERT führt eine Mailingliste die Sicherheitswarnungen zu verschiedenen Betriebssystemen und Anwendungen veröffentlicht. Sie können die Liste unter *http://www.us-cert.gov/cas/signup.html* abonnieren.

  - Bugtraq

    Die Mailingliste Bugtraq ist eine sog. full-disclosure Mailingliste. Auf ihr werden neu entdeckte Sicherheitsprobleme und zum Teil auch Patches zum Beheben der Fehler veröffentlicht. Sie können die Liste unter *http://www.securityfocus.com/archive* abonnieren.

- Beyond Linux From Scratch

  Das Buch „Beyond Linux From Scratch“ befasst sich mit der Installation einer Menge Software, die den Rahmen des LFS-Buches sprengen würde. Das BLFS-Projekt finden Sie unter *http://www.linuxfromscratch.org/blfs/*.

- LFS-Hints

  Die LFS-Hints sind eine Sammlung von nützlichen Anleitungen und Tipps, die von Freiwilligen aus der LFS-Gemeinschaft eingereicht wurden. Die Anleitungen sind verfügbar unter *http://www.linuxfromscratch.org/hints/list.html*.

- Mailinglisten

  Es gibt einige Mailinglisten, die Sie abonnieren können, wenn Sie mal Hilfe benötigen. Weitere Informationen finden Sie in Kapitel 1 - Mailinglisten.

- Das Linux Documentation Project

  Das Ziel des Linux Documentation Project ist es, in allen Fragen zu Linux zusammenzuarbeiten. Das LDP verfügt über jede Menge an HOWTOs, Anleitungen und Man-pages. Sie finden es unter *http://www.tldp.org/*.

Teil IV. Anhänge

Anhang A. Akronyme und Begriffe

ABI Application Binary Interface

ALFS Automated Linux From Scratch

ALSA Advanced Linux Sound Architecture

API Application Programming Interface

ASCII American Standard Code for Information Interchange

BIOS Basic Input/Output System

BLFS Beyond Linux From Scratch

BSD Berkeley Software Distribution

chroot change root

CMOS Complementary Metal Oxide Semiconductor

COS Class Of Service

CPU Central Processing Unit

CRC Cyclic Redundancy Check

CVS Concurrent Versions System

DHCP Dynamic Host Configuration Protocol

DNS Domain Name Service

EGA Enhanced Graphics Adapter

ELF Executable and Linkable Format

EOF End of File

EQN equation

EVMS Enterprise Volume Management System

ext2 second extended file system

ext3 third extended file system

FAQ Frequently Asked Questions

FHS Filesystem Hierarchy Standard

FIFO First-In, First Out

FQDN Fully Qualified Domain Name

FTP File Transfer Protocol

GB Gibabytes

GCC GNU Compiler Collection

GID Group Identifier

GMT Greenwich Mean Time
GPG GNU Privacy Guard
HTML Hypertext Markup Language
IDE Integrated Drive Electronics
IEEE Institute of Electrical and Electronic Engineers
IO Input/Output
IP Internet Protocol
IPC Inter-Process Communication
IRC Internet Relay Chat
ISO International Organization for Standardization
ISP Internet Service Provider
KB Kilobytes
LED Light Emitting Diode
LFS Linux From Scratch
LSB Linux Standard Base
MB Megabytes
MBR Master Boot Record
MD5 Message Digest 5
NIC Network Interface Card
NLS Native Language Support
NNTP Network News Transport Protocol
NPTL Native POSIX Threading Library
OSS Open Sound System
PCH Pre-Compiled Headers
PCRE Perl Compatible Regular Expression
PID Process Identifier
PLFS Pure Linux From Scratch
PTY pseudo terminal
QA Quality Assurance
QOS Quality Of Service
RAM Random Access Memory
RPC Remote Procedure Call
RTC Real Time Clock
SBU Standard Build Unit

SCO The Santa Cruz Operation

SGR Select Graphic Rendition

SHA1 Secure-Hash Algorithm 1

SMP Symmetric Multi-Processor

TLDP Das Linux Documentation Project

TFTP Trivial File Transfer Protocol

TLS Thread-Local Storage

UID User Identifier

umask user file-creation mask

USB Universal Serial Bus

UTC Coordinated Universal Time

UUID Universally Unique Identifier

VC Virtual Console

VGA Video Graphics Array

VT Virtual Terminal

Anhang B. Danksagungen

Wir möchten uns bei allen nachfolgenden Personen und Organisationen für ihr Mitwirken und die Beiträge zu Linux From Scratch bedanken.

- *Gerard Beekmans* <gerard@linuxfromscratch.org> – Gründer von Linux From Scratch, LFS-Projektbetreuer
- *Matthew Burgess* <matthew@linuxfromscratch.org> – LFS-Projektleiter, Release-Betreuer, Buchautor
- *Archaic* <archaic@linuxfromscratch.org> – LFS Buchautor, HLFS-Projektleiter, BLFS-Buchautor, Projektbetreuer von Hints and Patches
- *Nathan Coulson* <nathan@linuxfromscratch.org> – Betreuer der LFS Bootskripte
- *Bruce Dubbs* <bdubbs@linuxfromscratch.org> – BLFS-Projektleiter
- *Manuel Canales Esparcia* <manuel@linuxfromscratch.org> – LFS/BLFS/HLFS XML- und XSL-Betreuer
- *Jim Gifford* <jim@linuxfromscratch.org> – LFS Buchautor, Patches-Projekt
- *Jeremy Huntwork* <jhuntwork@linuxfromscratch.org> – ALFS-Betreuer, LFS Live-CD-Betreuer, LFS-Buchautor
- *Anderson Lizardo* <lizardo@linuxfromscratch.org> – Betreuer der Website-Skripte
- *Ryan Oliver* <ryan@linuxfromscratch.org> – LFS-Toolchain-Betreuer
- *James Robertson* <jwrober@linuxfromscratch.org> – Bugzilla-Betreuer
- *Tushar Teredesai* <tushar@linuxfromscratch.org> – BLFS-Buchautor, Betreuer des Hints und Patches Projekts
- Zahllose weitere Personen aus den verschiedenen LFS- und BLFS-Mailinglisten, die mit Vorschlägen, Tests und Fehlerberichten, Anleitungen und Installationserfahrungen zu diesem Buch beitragen.

Übersetzer

- *Manuel Canales Esparcia* <macana@macana-es.com> – Spanisches LFS-Übersetzerprojekt
- *Johan Lenglet* <johan@linuxfromscratch.org> – Französisches LFS-Übersetzerprojekt
- *Anderson Lizardo* <lizardo@linuxfromscratch.org> – Portugiesisches LFS-Übersetzerprojekt
- *Thomas Reitelbach* <tr@erdfunkstelle.de> – Deutsches LFS-Übersetzerprojekt

Betreuer der Softwarespiegel

Nordamerikanische Spiegel

- *Scott Kveton* <scott@osuosl.org> – lfs.oregonstate.edu
- *Mikhail Pastukhov* <miha@xuy.biz> – lfs.130th.net
- *William Astle* <lost@l-w.net> – ca.linuxfromscratch.org

- *Jeremy Polen* <jpolen@rackspace.com> – us2.linuxfromscratch.org
- *Tim Jackson* <tim@idge.net> – linuxfromscratch.idge.net
- *Jeremy Utley* <jeremy@linux-phreak.net> – lfs.linux-phreak.net

Südamerikanische Spiegel

- *Andres Meggiotto* <sysop@mesi.com.ar> – lfs.mesi.com.ar
- *Manuel Canales Esparcia* <manuel@linuxfromscratch.org> – lfsmirror.lfs-es.info
- *Eduardo B. Fonseca* <ebf@aedsolucoes.com.br> – br.linuxfromscratch.org

Europäische Spiegel

- *Barna Koczka* <barna@siker.hu> – hu.linuxfromscratch.org
- *UK Mirror Service* – linuxfromscratch.mirror.ac.uk
- *Martin Voss* <Martin.Voss@ada.de> – lfs.linux-matrix.net
- *Guido Passet* <guido@primerelay.net> – nl.linuxfromscratch.org
- *Bastiaan Jacques* <baafie@planet.nl> – lfs.pagefault.net
- *Roel Neefs* <lfs-mirror@linuxfromscratch.rave.org> – linuxfromscratch.rave.org
- *Justin Knierim* <justin@jrknierim.de> – www.lfs-matrix.de
- *Stephan Brendel* <stevie@stevie20.de> – lfs.netservice-neuss.de
- *Antonin Sprinzl* <Antonin.Sprinzl@tuwien.ac.at> – at.linuxfromscratch.org
- *Fredrik Danerklint* <fredan-lfs@fredan.org> – se.linuxfromscratch.org
- *Parisian sysadmins* <archive@doc.cs.univ-paris8.fr> – www2.fr.linuxfromscratch.org
- *Alexander Velin* <velin@zadnik.org> – bg.linuxfromscratch.org
- *Dirk Webster* <dirk@securewebservices.co.uk> – lfs.securewebservices.co.uk
- *Thomas Skyt* <thomas@sofagang.dk> – dk.linuxfromscratch.org
- *Simon Nicoll* <sime@dot-sime.com> – uk.linuxfromscratch.org

Asiatische Spiegel

- *Pui Yong* <pyng@spam.averse.net> – sg.linuxfromscratch.org
- *Stuart Harris* <stuart@althalus.me.uk> – lfs.mirror.intermedia.com.sg

Australische Spiegel

- *Jason Andrade* <jason@dstc.edu.au> – au.linuxfromscratch.org

Frühere Projektmitglieder

- *Christine Barczak* <theladyskye@linuxfromscratch.org> – LFS Buchautorin
- Timothy Bauscher
- Robert Briggs
- Ian Chilton
- *Jeroen Coumans* <jeroen@linuxfromscratch.org> – Website-Entwickler, Betreuer der FAQ
- Alex Groenewoud – LFS Technischer Autor
- Marc Heerdink
- Mark Hymers
- Seth W. Klein – Betreuer der FAQ
- *Nicholas Leippe* <nicholas@linuxfromscratch.org> – Wiki-Betreuer
- Simon Perreault
- *Scot Mc Pherson* <scot@linuxfromscratch.org> – LFS NNTP Gateway-Betreuer
- *Alexander Patrakov* <semzx@newmail.ru> – LFS Buchautor
- *Greg Schafer* <gschafer@zip.com.au> – LFS Technischer Autor
- Jesse Tie-Ten-Quee – LFS Technischer Autor
- *Jeremy Utley* <jeremy@linuxfromscratch.org> – LFS Buchautor, Bugzilla-Betreuer, Betreuer der LFS Bootskripte
- *Zack Winkles* <zwinkles@gmail.com> – LFS Buchautor

Ein besonderer Dank gilt all unseren Spendern

- *Dean Benson* <dean@vipersoft.co.uk> für etliche Geldspenden
- *Hagen Herrschaft* <hrx@hrxnet.de> für die Spende eines 2,2 GHz P4-Systems, welches nun unter dem Namen Lorien läuft
- *SEO Company Canada* unterstützt Open-Source-Projekte und verschiedene Linux-Distributionen
- *VA Software* die, im Namen von *Linux.com*, eine VA Linux 420 (ehem. StartX SP2) Workstation gespendet haben
- Mark Stone für die Spende von Belgarath, dem linuxfromscratch.org Server

Anhang C. Abhängigkeiten

Jedes in LFS installierte Paket verlässt sich zum Kompilieren und Installieren auf ein oder mehrere weitere Pakete. Manche Pakete haben sogar rekursive Abhängigkeiten. Das heißt, ein Paket A benötigt Paket B, welches wiederum Paket A voraussetzt. Diese z. T. recht komplizierten Abhängigkeiten begründen auch die besondere Installationsreihenfolge der Pakete in LFS. Der Zweck dieser Seite ist es, die Abhängigkeiten aller Pakete in LFS zu dokumentieren.

Für jedes installierte Paket listen wir hier drei Arten von Abhängigkeiten auf. Die erste Liste enthält Pakete, die zur Installation der fraglichen Software benötigt werden. Die zweite Liste enthält die Pakete, die zum korrekten Durchlaufen der Testsuite der fraglichen Software benötigt werden. Die dritte Liste enthält die LFS-Programme, die dieses fragliche Paket zur korrekten Installation voraussetzen (und zwar am endgültigen Installationsort fertig installiert!). In den meisten Fällen ist der Grund dafür, das diese Programme die Pfade zum fraglichen Paket fest in Skripten einbinden. Wenn Sie sich nicht an die in LFS vorgegebene Installationsreihenfolge halten, könnten diese Programm Pfade wie /tools/bin/[binärdatei] in ihren Skripten einbinden; dies wäre absolut nicht wünschenswert.

Autoconf

Installation ist abhängig von: Bash, Coreutils, Grep, M4, Make, Perl, Sed und Texinfo

Testsuite ist abhängig von: Automake, Diffutils, Findutils, GCC und Libtool

Muss installiert werden vor: Automake

Automake

Installation ist abhängig von: Autoconf, Bash, Coreutils, Gettext, Grep, M4, Make, Perl, Sed und Texinfo

Testsuite ist abhängig von: Binutils, Bison, Bzip2, DejaGNU, Diffutils, Expect, Findutils, Flex, GCC, Gettext, Gzip, Libtool und Tar. Kann auch noch einige weitere Pakete verwenden, die nicht mit LFS installiert werden.

Muss installiert werden vor: Keine

Bash

Installation ist abhängig von: Bash, Bison, Coreutils, Diffutils, GCC, Glibc, Grep, Make, Ncurses, Patch, Readline, Sed und Texinfo

Testsuite ist abhängig von: Diffutils und Gawk

Muss installiert werden vor: Keine

Berkeley DB

Installation ist abhängig von: Bash, Binutils, Coreutils, Diffutils, Gawk, GCC, Glibc, Grep, Make und Sed

Testsuite ist abhängig von: Wird nicht ausgeführt. Benötigt ein im fertigen System installiertes TCL.

Muss installiert werden vor: Keine

Binutils

Installation ist abhängig von: Bash, Binutils, Coreutils, Diffutils, GCC, Gettext, Glibc, Grep, Make, Perl, Sed und Texinfo

Testsuite ist abhängig von: DejaGNU und Expect

Muss installiert werden vor: Keine

Bison

Installation ist abhängig von: Bash, Binutils, Coreutils, GCC, Gettext, Glibc, Grep, M4, Make und Sed

Testsuite ist abhängig von: Diffutils und Findutils

Muss installiert werden vor: Flex, Kbd und Tar

Bzip2

Installation ist abhängig von: Bash, Binutils, Coreutils, Diffutils, GCC, Glibc, Make und Patch

Testsuite ist abhängig von: Keine

Muss installiert werden vor: Keine

Coreutils

Installation ist abhängig von: Bash, Binutils, Coreutils, GCC, Gettext, Glibc, Grep, Make, Patch, Perl, Sed und Texinfo

Testsuite ist abhängig von: Diffutils

Muss installiert werden vor: Bash, Diffutils, Findutils, Man-DB und Udev

DejaGNU

Installation ist abhängig von: Bash, Coreutils, Diffutils, GCC, Grep, Make und Sed

Testsuite ist abhängig von: Enthält keine Testsuite

Muss installiert werden vor: Keine

Diffutils

Installation ist abhängig von: Bash, Binutils, Coreutils, Diffutils, GCC, Gettext, Glibc, Grep, Make, Patch, Sed und Texinfo

Testsuite ist abhängig von: Enthält keine Testsuite

Muss installiert werden vor: Keine

Expect

Installation ist abhängig von: Bash, Binutils, Coreutils, Diffutils, GCC, Glibc, Grep, Make, Patch, Sed und Tcl

Testsuite ist abhängig von: Keine

Muss installiert werden vor: Keine

E2fsprogs

Installation ist abhängig von: Bash, Binutils, Coreutils, Gawk, GCC, Gettext, Glibc, Grep, Gzip, Make, Sed und Texinfo

Testsuite ist abhängig von: Diffutils

Muss installiert werden vor: Util-Linux

File

Installation ist abhängig von: Bash, Binutils, Coreutils, Diffutils, Gawk, GCC, Glibc, Grep, Make, Sed und Zlib

Testsuite ist abhängig von: Enthält keine Testsuite

Muss installiert werden vor: Keine

Findutils

Installation ist abhängig von: Bash, Binutils, Coreutils, GCC, Gettext, Glibc, Grep, Make, Sed und Texinfo

Testsuite ist abhängig von: DejaGNU, Diffutils und Expect

Muss installiert werden vor: Keine

Flex

Installation ist abhängig von: Bash, Binutils, Coreutils, GCC, Gettext, Glibc, Grep, M4, Make, Patch, Sed und Texinfo

Testsuite ist abhängig von: Bison und Gawk

Muss installiert werden vor: IPRoute2, Kbd und Man-DB

Gawk

Installation ist abhängig von: Bash, Binutils, Coreutils, GCC, Gettext, Glibc, Grep, Make, Patch, Sed und Texinfo

Testsuite ist abhängig von: Diffutils

Muss installiert werden vor: Keine

Gcc

Installation ist abhängig von: Bash, Binutils, Coreutils, Diffutils, Findutils, Gawk, GCC, Gettext, Glibc, Grep, Make, Patch, Perl, Sed, Tar und Texinfo

Testsuite ist abhängig von: DejaGNU und Expect

Muss installiert werden vor: Keine

Gettext

Installation ist abhängig von: Bash, Binutils, Coreutils, Gawk, GCC, Glibc, Grep, Make, Sed und Texinfo

Testsuite ist abhängig von: Diffutils, Perl und Tcl

Muss installiert werden vor: Automake

Glibc

Installation ist abhängig von: Bash, Binutils, Coreutils, Diffutils, Gawk, GCC, Gettext, Grep, Gzip, Make, Perl, Sed und Texinfo

Testsuite ist abhängig von: Keine

Muss installiert werden vor: Keine

Grep

Installation ist abhängig von: Bash, Binutils, Coreutils, Diffutils, GCC, Gettext, Glibc, Make, Patch, Sed und Texinfo

Testsuite ist abhängig von: Diffutils und Gawk

Muss installiert werden vor: Man-DB

Groff

Installation ist abhängig von: Bash, Binutils, Bison, Coreutils, Gawk, GCC, Glibc, Grep, Make, Patch, Sed und Texinfo

Testsuite ist abhängig von: Enthält keine Testsuite

Muss installiert werden vor: Man-DB und Perl

GRUB

Installation ist abhängig von: Bash, Binutils, Coreutils, Diffutils, GCC, Glibc, Grep, Make, Ncurses, Sed und Texinfo

Testsuite ist abhängig von: Keine

Muss installiert werden vor: Keine

Gzip

Installation ist abhängig von: Bash, Binutils, Coreutils, Diffutils, GCC, Glibc, Grep, Make, Patch, Sed und Texinfo

Testsuite ist abhängig von: Enthält keine Testsuite

Muss installiert werden vor: Man-DB

Iana-Etc

Installation ist abhängig von: Coreutils, Gawk und Make

Testsuite ist abhängig von: Enthält keine Testsuite

Muss installiert werden vor: Perl

Inetutils

Installation ist abhängig von: Bash, Binutils, Coreutils, GCC, Glibc, Grep, Make, Ncurses, Patch, Sed und Texinfo

Testsuite ist abhängig von: Enthält keine Testsuite

Muss installiert werden vor: Tar

IProute2

Installation ist abhängig von: Bash, Berkeley DB, Bison, Coreutils, Flex, GCC, Glibc, Make und Linux-Libc-Headers

Testsuite ist abhängig von: Enthält keine Testsuite

Muss installiert werden vor: Keine

Kbd

Installation ist abhängig von: Bash, Binutils, Bison, Coreutils, Flex, GCC, Gettext, Glibc, Gzip, Make, Patch und Sed

Testsuite ist abhängig von: Enthält keine Testsuite

Muss installiert werden vor: Keine

Less

Installation ist abhängig von: Bash, Binutils, Coreutils, Diffutils, GCC, Glibc, Grep, Make, Ncurses und Sed

Testsuite ist abhängig von: Enthält keine Testsuite

Muss installiert werden vor: Keine

Libtool

Installation ist abhängig von: Bash, Binutils, Coreutils, Diffutils, Gawk, GCC, Glibc, Grep, Make, Sed und Texinfo

Testsuite ist abhängig von: Findutils

Muss installiert werden vor: Keine

Linux-Kernel

Installation ist abhängig von: Bash, Binutils, Coreutils, Diffutils, Findutils, GCC, Glibc, Grep, Gzip, Make, Module-Init-Tools, Ncurses und Sed

Testsuite ist abhängig von: Enthält keine Testsuite

Muss installiert werden vor: Keine

M4

Installation ist abhängig von: Bash, Binutils, Coreutils, GCC, Glibc, Grep, Make und Sed

Testsuite ist abhängig von: Diffutils

Muss installiert werden vor: Autoconf und Bison

Man-DB

Installation ist abhängig von: Bash, Berkeley DB, Binutils, Bzip2, Coreutils, Flex, GCC, Gettext, Glibc, Grep, Groff, Gzip, Less, Make und Sed

Testsuite ist abhängig von: Wird nicht ausgeführt. Benötigt das Testsuite-Paket von Man-DB.

Muss installiert werden vor: Keine

Make

Installation ist abhängig von: Bash, Binutils, Coreutils, Diffutils, GCC, Gettext, Glibc, Grep, Make, Sed und Texinfo

Testsuite ist abhängig von: Perl

Muss installiert werden vor: Keine

Mktemp

Installation ist abhängig von: Bash, Binutils, Coreutils, GCC, Glibc, Grep, Patch und Sed

Testsuite ist abhängig von: Enthält keine Testsuite

Muss installiert werden vor: Keine

Module-Init-Tools

Installation ist abhängig von: Bash, Binutils, Coreutils, GCC, Glibc, Grep, Make, Sed und Zlib

Testsuite ist abhängig von: File, Findutils und Gawk

Muss installiert werden vor: Keine

Ncurses

Installation ist abhängig von: Bash, Binutils, Coreutils, Diffutils, Gawk, GCC, Glibc, Grep, Make, Patch und Sed

Testsuite ist abhängig von: Enthält keine Testsuite

Muss installiert werden vor: Bash, GRUB, Inetutils, Less, Procps, Psmisc, Readline, Texinfo, Util-Linux und Vim

Patch

Installation ist abhängig von: Bash, Binutils, Coreutils, GCC, Glibc, Grep, Make und Sed

Testsuite ist abhängig von: Enthält keine Testsuite

Muss installiert werden vor: Keine

Perl

Installation ist abhängig von: Bash, Berkeley DB, Binutils, Coreutils, Gawk, GCC, Glibc, Grep, Groff, Make und Sed

Testsuite ist abhängig von: Iana-Etc und Procps

Muss installiert werden vor: Autoconf

Procps

Installation ist abhängig von: Bash, Binutils, Coreutils, GCC, Glibc, Make und Ncurses

Testsuite ist abhängig von: Enthält keine Testsuite

Muss installiert werden vor: Keine

Psmisc

Installation ist abhängig von: Bash, Coreutils, GCC, Gettext, Glibc, Grep, Make, Ncurses und Sed

Testsuite ist abhängig von: Enthält keine Testsuite

Muss installiert werden vor: Keine

Readline

Installation ist abhängig von: Bash, Binutils, Coreutils, GCC, Glibc, Grep, Make, Ncurses, Patch, Sed und Texinfo

Testsuite ist abhängig von: Enthält keine Testsuite

Muss installiert werden vor: Bash

Sed

Installation ist abhängig von: Bash, Binutils, Coreutils, GCC, Gettext, Glibc, Grep, Make, Sed und Texinfo

Testsuite ist abhängig von: Diffutils und Gawk

Muss installiert werden vor: E2fsprogs, File, Libtool und Shadow

Shadow

Installation ist abhängig von: Bash, Binutils, Coreutils, Diffutils, Findutils, Gawk, GCC, Gettext, Glibc, Grep, Make und Sed

Testsuite ist abhängig von: Enthält keine Testsuite

Muss installiert werden vor: Keine

Sysklogd

Installation ist abhängig von: Binutils, Coreutils, GCC, Glibc, Make und Patch

Testsuite ist abhängig von: Enthält keine Testsuite

Muss installiert werden vor: Keine

Sysvinit

Installation ist abhängig von: Binutils, Coreutils, GCC, Glibc, Make und Sed

Testsuite ist abhängig von: Enthält keine Testsuite

Muss installiert werden vor: Keine

Tar

Installation ist abhängig von: Bash, Binutils, Bison, Coreutils, GCC, Gettext, Glibc, Grep, Inetutils, Make, Patch, Sed und Texinfo

Testsuite ist abhängig von: Diffutils, Findutils und Gawk

Muss installiert werden vor: Keine

Tcl

Installation ist abhängig von: Bash, Binutils, Coreutils, Diffutils, GCC, Glibc, Grep, Make und Sed

Testsuite ist abhängig von: Keine

Muss installiert werden vor: Keine

Texinfo

Installation ist abhängig von: Bash, Binutils, Coreutils, GCC, Gettext, Glibc, Grep, Make, Ncurses, Patch und Sed

Testsuite ist abhängig von: Keine

Muss installiert werden vor: Keine

Udev

Installation ist abhängig von: Binutils, Coreutils, GCC, Glibc und Make

Testsuite ist abhängig von: Findutils, Perl und Sed

Muss installiert werden vor: Keine

Util-Linux

Installation ist abhängig von: Bash, Binutils, Coreutils, E2fprogs, GCC, Gettext, Glibc, Grep, Make, Ncurses, Patch, Sed und Zlib

Testsuite ist abhängig von: Enthält keine Testsuite

Muss installiert werden vor: Keine

Vim

Installation ist abhängig von: Bash, Binutils, Coreutils, Diffutils, GCC, Glibc, Grep, Make, Ncurses und Sed

Testsuite ist abhängig von: Keine

Muss installiert werden vor: Keine

Zlib

Installation ist abhängig von: Bash, Binutils, Coreutils, GCC, Glibc, Grep, Make und Sed

Testsuite ist abhängig von: Keine

Muss installiert werden vor: File, Module-Init-Tools und Util-Linux

Stichwortverzeichnis

Pakete

Programme

Bibliotheken

Skripte

Sonstige